



UNIVERSITAS INDONESIA

**ARSITEKTUR *EVENT-DRIVEN* DAN *BIG DATA* UNTUK SERVIS
APLIKASI KOTA CERDAS MAHONI**

SKRIPSI

MOHAMMAD RISWANDA ALIFARAHMAN	1906293171
MUHAMMAD FATHAN MUTHAHHARI	1906293190
NATASYA ZAHRA	1906293253

**FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JUNI 2023**



UNIVERSITAS INDONESIA

**ARSITEKTUR *EVENT-DRIVEN* DAN *BIG DATA* UNTUK SERVIS
APLIKASI KOTA CERDAS MAHONI**

SKRIPSI

**Diajukan sebagai salah satu syarat memperoleh gelar Sarjana Ilmu
Komputer**

MOHAMMAD RISWANDA ALIFARAHMAN	1906293171
MUHAMMAD FATHAN MUTHAHHARI	1906293190
NATASYA ZAHRA	1906293253

**FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JUNI 2023**

HALAMAN PERNYATAAN ORISINALITAS

**Skripsi ini adalah hasil karya kami sendiri, dan
semua sumber baik yang dikutip maupun
dirujuk telah kami nyatakan dengan benar.**

Nama Penulis 1 : Mohammad Riswanda Alifarahman

NPM Penulis 1 : 1906293171

Tanda Tangan Penulis 1



Nama Penulis 2 : Muhammad Fathan Muthahhari

NPM Penulis 2 : 1906293190

Tanda Tangan Penulis 2



Nama Penulis 3 : Natasya Zahra

NPM Penulis 3 : 1906293253

Tanda Tangan Penulis 3



HALAMAN PENGESAHAN

Skripsi ini diajukan oleh:

Penulis 1

Nama : Mohammad Riswanda Alifarahman
NPM : 1906293171
Program Studi : Ilmu Komputer

Penulis 2

Nama : Muhammad Fathan Muthahhari
NPM : 1906293190
Program Studi : Ilmu Komputer

Penulis 3

Nama : Natasya Zahra
NPM : 1906293253
Program Studi : Ilmu Komputer

Judul Skripsi : Arsitektur *Event-Driven* dan *Big Data* untuk Servis Aplikasi
Kota Cerdas Mahoni

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memeroleh gelar Sarjana pada Program Studi Ilmu Komputer, Fakultas Ilmu Komputer, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing 1 : Muhammad Hafizhuddin Hilman, S.Kom., M.Kom., Ph.D. ()
Pembimbing 2 : Ari Wibisono, S.Kom., M.Kom. ()
Penguji 1 : ()
Penguji 2 : ()
Ditetapkan di : Depok
Tanggal :

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Allah SWT karena berkat rahmat dan karunia-Nya penulis dapat mengerjakan dan menyelesaikan tugas akhir dengan judul ini. Penyusunan tugas akhir ini dilakukan untuk memenuhi salah satu persyaratan memperoleh gelar Sarjana Ilmu Komputer pada Fakultas Ilmu Komputer Universitas Indonesia. Penulis menyadari bahwa selama penulisan tugas akhir ini, tanpa bantuan dari berbagai pihak akan sulit bagi penulis untuk menyelesaikan tugas akhir ini. Oleh karena itu, penulis ingin menyampaikan rasa terima kasih kepada:

1. Orang tua dan keluarga penulis yang selalu mendoakan dan mendukung penulis dari awal perkuliahan sampai penulisan tugas akhir ini
2. Muhammad Hafizhuddin Hilman, S.Kom., M.Kom., Ph.D. sebagai dosen pembimbing pertama tugas akhir ini yang sudah menyempatkan waktu, tenaga dan pemikirannya agar penulis dapat menyelesaikan tugas akhir ini.
3. Ari Wibisono, S.Kom., M.Kom. sebagai dosen pembimbing kedua tugas akhir ini yang sudah menyempatkan waktu, tenaga dan pemikirannya agar penulis dapat menyelesaikan tugas akhir ini.
4. Seluruh dosen Fasilkom UI yang sudah memberikan ilmu kepada penulis sehingga penulis dapat mengerjakan tugas akhir ini dengan baik.
5. Pihak Jakarta Smart City dan Dinas Perhubungan DKI Jakarta yang sudah menyempatkan waktu terlibat dalam tugas akhir ini.
6. Teman-teman penulis yang sudah memberikan dukungan dan menemani dalam menyelesaikan tugas akhir ini.

Akhir kata kami berharap Allah SWT membalaik kebaikan semua pihak yang telah mendukung, mendoakan, dan membantu penulis dalam menyelesaikan tugas akhir ini. Semoga tugas akhir ini dapat bermanfaat dan memberikan sumbangsih untuk perkembangan ilmu pengetahuan.

Depok, 8 Juni 2023

Tim Penulis

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIK

Sebagai sivitas akademik Universitas Indonesia, kami yang bertanda tangan di bawah ini:

Nama : Mohammad Riswanda Alifarahman
Muhammad Fathan Muthahhari
Natasya Zahra
NPM : 1906293171
1906293190
1906293253
Program Studi : Ilmu Komputer
Fakultas : Ilmu Komputer
Jenis Karya : Skripsi

demi pengembangkan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (Non-exclusive Royalty Free Right)** atas karya ilmiah saya berjudul

“Arsitektur *Event-Driven* dan *Big Data* untuk Servis Aplikasi Kota Cerdas Mahoni”

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/format-kan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyataan ini saya buat dengan sebenarnya

Dibuat di : Depok
Pada tanggal : 8 Juni 2023

Yang menyatakan,

Penulis 1



(Mohammad Riswanda
Alifarahman)

Penulis 2



(Muhammad Fathan
Muthahhari)

Penulis 3



(Natasya Zahra)

ABSTRAK

Nama	:	Mohammad Riswanda Alifarahman Muhammad Fathan Muthahhari Natasya Zahra
Program Studi	:	Ilmu Komputer
Judul	:	Arsitektur <i>Event-Driven</i> dan <i>Big Data</i> untuk Servis Aplikasi Kota Cerdas Mahoni
Pembimbing	:	Muhammad Hafizhuddin Hilman, S.Kom., M.Kom., Ph.D. Ari Wibisono, S.Kom., M.Kom.

Tingginya jumlah kendaraan bermotor di Indonesia memiliki dampak kepada kualitas udara. Aplikasi Mahoni merupakan upaya solusi dari permasalahan tersebut dengan membawa konsep kota cerdas. Penulis melakukan pengembangan arsitektur *microservice* yang melayani fitur pada aplikasi Mahoni yaitu servis kualitas udara, perjalanan, dan penukaran poin menjadi kupon sesuai dengan kebutuhan pengguna. Aplikasi Mahoni dikembangkan dengan menggunakan arsitektur *event-driven* agar dapat mencatat beragam data yang berasal dari sensor udara dan aktivitas pengguna secara *real-time*. Kafka digunakan sebagai *message broker* untuk mendapatkan *throughput* yang tinggi dan mempermudah integrasi dengan komponen *big data* yang memerlukan data *stream* untuk melakukan *stream processing* dan *real-time analytics* melalui *change data capture* dengan bantuan Debezium dan Kafka Connect. Data *stream* diolah menjadi keluaran yang dibutuhkan seperti visualisasi data menggunakan *dashboard*. Untuk mencapai hal tersebut, arsitektur Kappa diimplementasikan untuk membangun arsitektur *big data* yang sederhana, *scalable*, dan *reliable*. Arsitektur *big data* pada penelitian ini terdiri dari beberapa komponen yaitu Flink, Cassandra, InfluxDB, dan Grafana. Keterhubungan implementasi keseluruhan arsitektur pada penelitian ini diuji dengan melakukan end-to-end testing. Hasil dari pengujian tersebut menunjukkan bahwa keseluruhan komponen sistem aplikasi Mahoni terhubung dengan baik dalam memenuhi kebutuhan pengguna. Komponen arsitektur *event-driven* juga dibuktikan dapat mengatasi data *stream* dengan *throughput* tinggi dan bersifat *loosely-coupled* sehingga integrasi komponen baru pada sistem lebih mudah. Komponen arsitektur *big data* juga dibuktikan dapat mengatasi

pertumbuhan data dengan melakukan *scaling* pada Flink sehingga menghasilkan sistem yang *reliable*.

Kata kunci:

Arsitektur *event-driven*, Kafka, *change data capture*, *big data*, Debezium, Kafka Connect, *microservice*, arsitektur Kappa, Flink, *stream processing*, *scalable*, *reliable*, *dashboard*

ABSTRACT

Names	:	Mohammad Riswanda Alifarahman Muhammad Fathan Muthahhari Natasya Zahra
Study Program	:	Computer Science
Title	:	Event-Driven Architecture and Big Data for Mahoni Smart City Application Services
Counsellors	:	Muhammad Hafizhuddin Hilman, S.Kom., M.Kom., Ph.D. Ari Wibisono, S.Kom., M.Kom.

The high number of motorized vehicles in Indonesia has an impact on air quality. Mahoni application is an attempt to solve the problem by bringing the concept of smart city. The author develops a microservice architecture that serves features in the Mahoni application, namely air quality services, travel, and redemption of points into coupons according to user needs. Mahoni application is developed using event-driven architecture in order to record various data from air sensors and user activities in real-time. Kafka is used as a message broker to get high throughput and facilitate integration with big data components that require data streams to perform stream processing and real-time analytics through change data capture with the help of Debezium and Kafka Connect. Stream data is processed into the required output such as data visualization using dashboards. To achieve this, Kappa architecture is implemented to build a simple, scalable, and reliable big data architecture. The big data architecture in this research consists of several components, namely Flink, Cassandra, InfluxDB, and Grafana. The connectedness of the implementation of the entire architecture in this study was tested by conducting end-to-end testing. The results of the test show that all components of the Mahoni application system are well connected in meeting user needs. The event-driven architecture component is also proven to be able to cope with high-throughput data streams and is loosely-coupled so that the integration of new components in the system is easier. The big data architecture component is also proven to be able to cope with data growth by scaling Flink to produce a reliable system.

Keywords:

Event-driven architecture, Kafka, change data capture, big data, Debezium, Kafka Connect, microservice, Kappa architecture, Flink, stream processing, scalable, reliable, dashboard

DAFTAR ISI

HALAMAN PERNYATAAN ORISINALITAS	iii
HALAMAN PENGESAHAN	iv
KATA PENGANTAR.....	v
HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIK	vii
ABSTRAK	viii
ABSTRACT	x
DAFTAR ISI	xi
DAFTAR GAMBAR.....	xvii
DAFTAR TABEL	xx
DAFTAR KODE	xxi
DAFTAR LAMPIRAN	xxii
BAB 1 PENDAHULUAN.....	1
1.1 Latar Belakang	1
1.2 Pertanyaan Penelitian.....	4
1.3 Tujuan Penelitian	5
1.4 Batasan Penelitian.....	5
1.5 Sistematika Penulisan	6
BAB 2 TINJAUAN PUSTAKA.....	7
2.1 Konsep Aplikasi.....	7
2.1.1 Kota Cerdas	7
2.1.2 Indeks Kualitas Udara	9
2.1.3 Sistem Penghargaan Poin	10
2.1.3.1 Implementasi Terkait	11
2.1.3.2 Penelitian Terkait.....	11
2.2 Terminologi Metodologi.....	14
2.2.1 Kano <i>Method</i>	14
2.2.2 Scrumban.....	18
2.3 Arsitektur Sistem dan Data	19
2.3.1 <i>Microservice</i>	19
2.3.2 <i>Event-Driven</i>	20

2.3.2.1 Penelitian Terkait.....	21
2.3.3 <i>Big Data</i>	22
2.3.3.1 Arsitektur Lambda.....	23
2.3.3.2 Arsitektur Kappa	23
2.4 Apache Kafka	24
2.4.1 Kafka <i>Topic</i> dan Partisi.....	25
2.4.2 Kafka <i>Consumer</i> dan Kafka <i>Producer</i>	27
2.4.3 Serialisasi dan Kafka <i>Schema Registry</i>	28
2.4.4 KTable	29
2.5 <i>Stream Processing</i>	30
2.5.1 Windowing	30
2.5.2 <i>Stream Join</i>	33
2.5.3 Apache Flink	34
2.6 Penyimpanan Data	36
2.6.1 <i>Relational Database</i>	36
2.6.2 <i>Non-Relational Database (NoSQL)</i>	37
2.6.2.1 InfluxDB	37
2.6.2.2 Cassandra.....	38
2.6.3 <i>Data Warehouse</i>	38
2.7 Change Data Capture (CDC)	40
BAB 3 METODE DAN METODOLOGI UMUM	42
3.1 Pembagian Metode Penyelesaian Masalah	42
3.2 Metodologi Umum.....	43
3.2.1 Tahapan Penelitian	44
3.2.2 Gambaran Umum	45
3.3 Skenario dan Matriks Evaluasi Umum	46
3.3.1 Skenario Evaluasi Umum.....	46
3.3.2 Metrik Evaluasi Umum	47
BAB 4 PENGEMBANGAN SERVIS-SERVIS MAHONI	49
4.1 Metode dan Metodologi	49
4.1.1 Tahapan Penelitian	49
4.1.2 Metodologi Pengembangan.....	50
4.1.3 Skenario Pengujian.....	51
4.1.4 Matriks Evaluasi.....	52

4.2 Praperancangan Servis	53
4.2.1 Analisis Domain Aplikasi	53
4.2.2 Identifikasi <i>Stakeholders</i>	54
4.2.3 Cerita Pengguna dan Sistem.....	54
4.2.4 Pembuatan Purwarupa.....	57
4.2.5 Elitisasi Kebutuhan Fungsional.....	58
4.2.5.1 Analisis <i>Task</i>	58
4.2.5.2 Wawancara	60
4.2.6 Elitisasi Kebutuhan Non-Fungsional	63
4.2.7 Elitisasi Batasan Kebutuhan.....	64
4.2.8 Pengelompokan Kebutuhan.....	64
4.2.8.1 Desain dan Responden.....	65
4.2.8.2 Analisis Data dan Hasil	66
4.3 Perancangan Servis	68
4.3.1 Pemetaan Kebutuhan dan Sistem	68
4.3.2 Pemilihan Teknologi	71
4.3.3 Perancangan Servis Pengguna.....	72
4.3.4 Perancangan Servis Kualitas Udara	73
4.3.5 Perancangan Servis Perjalanan.....	75
4.3.6 Perancangan Servis Kupon dan Mitra Usaha	77
4.4 Pengembangan Servis	81
4.4.1 Pengembangan Servis Pengguna.....	83
4.4.2 Pengembangan Servis Kualitas Udara	84
4.4.3 Pengembangan Servis Perjalanan.....	85
4.4.4 Pengembangan Servis Kupon dan Mitra Usaha	86
4.5 Evaluasi Servis.....	89
4.5.1 <i>Unit Testing</i>	89
4.5.2 <i>Integration Testing</i>	93
4.5.3 <i>End-To-End Testing</i>	95
4.6 Kesimpulan Pengembangan Servis-Servis Mahoni	96
BAB 5 ARSITEKTUR <i>EVENT-DRIVEN MAHONI</i>	97
5.1 Metodologi Penelitian.....	97
5.1.1 Tahapan Penelitian	97
5.1.2 Skenario Pengujian.....	98

5.1.3 Matriks Evaluasi.....	99
5.2 Perancangan Arsitektur <i>Event-Driven</i>	100
5.2.1 Pemilihan <i>Framework</i>	100
5.2.1.1 <i>Message Broker</i>	100
5.2.1.2 <i>Change Data Capture</i>	102
5.2.2 Rancangan Arsitektur <i>Event-Driven</i>	102
5.2.2.1 Kafka <i>Cluster</i>	103
5.2.2.2 Kafka <i>Topic</i>	103
5.2.2.3 <i>Domain Event</i> dan <i>Schema</i>	105
5.2.2.4 Servis Web dan <i>Database</i>	105
5.2.2.5 Konsistensi	106
5.2.2.6 <i>Change Data Capture</i>	109
5.2.2.7 Arsitektur <i>Event-Driven</i>	109
5.3 Implementasi Arsitektur <i>Event-Driven</i>	110
5.3.1 Infrastruktur pada <i>Docker Virtual Machine</i>	111
5.3.1.1 Kafka <i>Cluster</i> dan <i>Schema Registry</i>	111
5.3.1.2 <i>Change Data Capture</i>	112
5.3.1.2.1 Konfigurasi Replikasi dan WAL pada PostgreSQL	112
5.3.1.2.2 Debezium dan <i>Source Connector</i>	113
5.3.1.2.3 Kafka Connect dan <i>Sink Connector</i>	115
5.3.2 Servis Web	117
5.3.2.1 Kafka <i>Producer</i> dan <i>Consumer</i>	118
5.3.2.2 KTable	120
5.3.2.3 Deployment	122
5.4 Evaluasi Arsitektur <i>Event-Driven</i>	123
5.4.1 Evaluasi Performa	124
5.4.2 Evaluasi Sifat <i>Loosely-Coupled</i>	127
5.5 Kesimpulan Arsitektur <i>Event-Driven</i>	129
BAB 6 ARSITEKTUR <i>BIG DATA</i> MAHONI.....	131
6.1 Metodologi Penelitian.....	131
6.1.1 Tahapan Penelitian	131
6.1.2 Skenario Pengujian.....	132
6.1.3 Matriks Evaluasi.....	133
6.2 Praperancangan Arsitektur Big Data	133

6.2.1 Pengumpulan Data Kebutuhan Pemerintah.....	134
6.2.1.1 Hasil Wawancara dengan Jakarta Smart City.....	134
6.2.1.2 Hasil Wawancara dengan Dinas Perhubungan DKI Jakarta	135
6.2.2 Pembuatan <i>User Story Dashboard</i>	135
6.3 Perancangan Arsitektur <i>Big Data</i>	136
6.3.1 Rancangan Arsitektur <i>Big Data</i>	137
6.3.1.1 <i>Acquisition Layer</i>	138
6.3.1.2 <i>Streaming Layer</i>	138
6.3.1.3 <i>Serve Layer</i> dan <i>Visualization Layer</i>	139
6.3.1.4 <i>Data Warehouse</i>	139
6.3.2 Pemilihan <i>Framework</i>	140
6.3.2.1 Apache Flink	140
6.3.2.2 Apache Cassandra.....	142
6.3.2.3 InfluxDB	143
6.3.3 Penyimpanan Data.....	144
6.3.3.1 Rancangan <i>Database Cassandra</i>	144
6.3.3.2 Rancangan <i>Database InfluxDB</i>	145
6.3.3.3 Rancangan <i>Data Warehouse</i>	147
6.4 Implementasi Arsitektur <i>Big Data</i>	148
6.4.1 Implementasi <i>Acquisition Layer</i>	148
6.4.2 Implementasi <i>Streaming layer</i>	150
6.4.2.1 Pembuatan <i>Air Quality Job</i>	150
6.4.2.2 Pembuatan <i>Trip Job</i> dan <i>Merchant & Voucher Job</i>	154
6.4.2.3 <i>Deployment Flink</i>	156
6.4.3 Implementasi <i>Serve Layer</i>	158
6.4.4 Implementasi <i>Visualization Layer</i>	158
6.4.5 Implementasi <i>Data Warehouse</i>	159
6.5 Evaluasi Arsitektur <i>Big Data</i>	161
6.5.1 Evaluasi Fungsionalitas.....	162
6.5.2 Evaluasi <i>Dashboard</i>	164
6.5.3 Evaluasi <i>Scalability</i> dan <i>Reliability</i> Arsitektur	170
6.6 Kesimpulan Arsitektur <i>Big Data</i>	174
BAB 7 EVALUASI DAN KESIMPULAN UMUM.....	176
7.1 Evaluasi Umum.....	176

7.1.1 Evaluasi Kuantitatif.....	176
7.1.2 Evaluasi Kualitatif.....	179
7.2 Kesimpulan Umum	182
7.3 Saran	183
DAFTAR PUSTAKA.....	186
LAMPIRAN	193

DAFTAR GAMBAR

Gambar 1.1 Perkembangan Jumlah Kendaraan Bermotor menurut Jenis (Unit) Tahun 2019-2021	2
Gambar 2.1 SNI 37122:2019 Sebagai Indikator Kota Cerdas	8
Gambar 2.2 Diagram Kano	15
Gambar 2.3 Diagram Representasi Kategori Kano.....	17
Gambar 2.4 Perbedaan Topologi Arsitektur Monolitik dan <i>Microservice</i>	20
Gambar 2.5 Arsitektur Lambda	23
Gambar 2.6 Arsitektur Kappa	24
Gambar 2.7 Arsitektur Kafka <i>Cluster</i>	25
Gambar 2.8 Representasi <i>Log</i> pada Sebuah Kafka <i>Topic</i> dengan Beberapa Partisi	26
Gambar 2.9 <i>Key-Based Retention Type Topic</i>	26
Gambar 2.10 Pembagian Beban antar <i>Consumer Group</i> yang <i>Subscribe</i> ke Sebuah <i>Topic</i>	28
Gambar 2.11 Alur Pengiriman <i>Event</i> oleh Kafka <i>Producer</i>	28
Gambar 2.12 Alur Serialisasi Menggunakan <i>Schema Registry</i>	29
Gambar 2.13 <i>State Store</i> di Kafka Streams API Menggunakan KTable	30
Gambar 2.14 <i>Tumbling Window</i>	31
Gambar 2.15 <i>Sliding Window</i>	32
Gambar 2.16 <i>Session Window</i>	32
Gambar 2.17 Arsitektur Flink	35
Gambar 2.18 Arsitektur Cassandra	38
Gambar 2.19 <i>Star Schema</i>	39
Gambar 2.20 <i>Change Data Capture</i>	40
Gambar 3.1 Bagan Tahapan Penelitian Secara Umum	44
Gambar 3.2 <i>High-Level</i> Arsitektur Mahoni	45
Gambar 4.1 Tahapan Penelitian Pengembangan Servis-Servis Mahoni.....	49
Gambar 4.2 Papan Scrumban.....	50
Gambar 4.3 Proses <i>Requirement Gathering</i>	53
Gambar 4.4 Proses Bisnis Mahoni	56
Gambar 4.5 Purwarupa Fidelitas Tinggi Aplikasi Mahoni	57
Gambar 4.6 Pemetaan Hasil Kuesioner pada Kano Diagram	67

Gambar 4.7 Use Case Diagram Mahoni.....	69
Gambar 4.8 Arsitektur Microservice Mahoni	70
Gambar 4.9 Activity Diagram Mengubah Profil	72
Gambar 4.10 ER Diagram Servis Pengguna.....	73
Gambar 4.11 Activity Diagram Lihat Informasi Kualitas Udara	74
Gambar 4.12 Activity Diagram Pencarian AQI dengan Lokasi.....	74
Gambar 4.13 ER Diagram Servis Kualitas Udara	75
Gambar 4.14 Activity Diagram Scan QR Code Transportasi	76
Gambar 4.15 Activity Diagram Melihat Riwayat Perjalanan	76
Gambar 4.16 ER Diagram Servis Perjalanan.....	76
Gambar 4.17 Activity Diagram Lihat Poin dan Kupon.....	78
Gambar 4.18 Activity Diagram Tukar Poin	78
Gambar 4.19 Activity Diagram Lihat Riwayat Redeem Kupon.....	78
Gambar 4.20 Activity Diagram Ubah Informasi Merchant	79
Gambar 4.21 Activity Diagram Mengelola Kupon	79
Gambar 4.22 Activity Diagram Subflow S1-S3 Mengelola Kupon.....	80
Gambar 4.23 ER Diagram Servis Kupon dan Mitra Usaha.....	81
Gambar 4.24 Layer Setiap Servis	82
Gambar 4.25 Struktur Pohon Servis Pengguna.....	83
Gambar 4.26 Struktur Pohon Servis Kualitas Udara	84
Gambar 4.27 Struktur Pohon Servis Perjalanan	85
Gambar 4.28 Struktur Pohon Servis Kupon dan Mitra Usaha: <i>Auth</i> dan <i>Merchant</i>	86
Gambar 4.29 Struktur Pohon Servis Kupon dan Mitra Usaha: <i>Voucher</i>	87
Gambar 5.1 Tahapan Penelitian Arsitektur <i>Event-Driven</i>	97
Gambar 5.2 <i>Trip Flow</i> untuk Memastikan Konsistensi Nilai Poin <i>User</i>	106
Gambar 5.3 <i>Redeem Voucher Flow</i> untuk Memastikan Konsistensi Nilai Poin <i>User</i> .107	107
Gambar 5.4 Arsitektur Change Data Capture dengan Debezium dan Kafka Connect 109	109
Gambar 5.5 Arsitektur <i>event-driven</i> aplikasi Mahoni.....	110
Gambar 5.6 Tampilan Kafka UI untuk Memonitor Kafka <i>cluster</i>	112
Gambar 5.7 Daftar <i>Topic</i> yang Dibuat oleh <i>Connector Debezium</i>	115
Gambar 5.8 Konfigurasi <i>Container</i> pada Google Cloud Platform <i>Compute Engine</i> ..123	123
Gambar 5.9 Detail Hasil Evaluasi Performa Skenario Pertama.....	124
Gambar 5.10 Detail Hasil Evaluasi Performa Skenario Kedua	125
Gambar 5.11 Detail Hasil Evaluasi Performa Skenario Ketiga	125

Gambar 5.15 Response Servis Loyalitas.....	129
Gambar 6.1 Tahapan Penelitian Arsitektur Big Data	131
Gambar 6.2 Arsitektur <i>Big Data</i>	137
Gambar 6.3 Skema Diagram Cassandra	145
Gambar 6.4 Skema Diagram InfluxDB	146
Gambar 6.5 Skema Star <i>Data Warehouse</i>	147
Gambar 6.6 Alur <i>Air Quality Job</i>	150
Gambar 6.7 Proses <i>Enrichment</i> Data pada <i>Air Quality Job</i>	153
Gambar 6.8 Alur <i>Trip Job</i> dan <i>Voucher & Merchant Job</i>	155
Gambar 6.9 Proses <i>Enrichment Trip Job</i> dan <i>Voucher & Merchant Job</i>	155
Gambar 6.10 <i>Deployment Flink</i>	157
Gambar 6.11 Tabel pada BigQuery	161
Gambar 6.12 Kompilasi Koneksi Topik Kafka	162
Gambar 6.14 <i>Connection Grafana ke InfluxDB</i>	164
Gambar 6.15 <i>Dashboard Grafik Kualitas Udara</i>	165
Gambar 6.16 <i>Dashboard Nilai Maksimum, Minimum, dan Rata-Rata Kualitas Udara</i>	166
Gambar 6.17 <i>Dashboard Scan In</i> dan <i>Scan Out</i>	166
Gambar 6.18 <i>Dashboard Perjalanan berdasarkan Jenis Kelamin</i>	167
Gambar 6.19 <i>Dashboard Status Perjalanan</i>	167
Gambar 6.20 Laporan Bulanan Penggunaan Transportasi Umum	168
Gambar 6.21 <i>Dashboard Penukaran Voucher berdasarkan Jenis Kelamin</i>	169
Gambar 6.22 Dashborad Pie Chart Penukaran Voucher.....	169
Gambar 6.23 <i>Dashboard Total Pembelian Voucher</i>	170
Gambar 6.24 Alur <i>Job</i> di <i>Web Interface Flink</i>	172
Gambar 6.25 Kegagalan <i>JobManager Flink</i>	173
Gambar 7.1 <i>Dashboard Hasil Scan In</i>	179
Gambar 7.2 <i>Dashboard Hasil Scan Out</i>	180
Gambar 7.3 <i>Dashboard Hasil Redeemed Voucher</i>	181
Gambar 7.4 <i>Dashboard Hasil Kualitas Udara</i>	181
Gambar 7.5 Data pada Cassandra	182
Gambar 7.6 Data pada Google BigQuery	182

DAFTAR TABEL

Tabel 2.1 Kategori Tingkat Kualitas Udara berdasarkan AQI	10
Tabel 2.2 Penelitian Terdahulu Implementasi Sistem Penghargaan Poin	12
Tabel 2.3 Tabel Evaluasi Kano.....	16
Tabel 2.4 Elemen Dasar Penyusun Scrumban	18
Tabel 3.1 Pembagian Ruang Lingkup.....	42
Tabel 3.2 Skenario Pengujian <i>End-to-End</i>	47
Tabel 4.1 Skenario Pengujian Pengembangan Servis-Servis Mahoni	51
Tabel 4.2 Hasil Analisis <i>Task</i> Aplikasi Kualitas Udara	59
Tabel 4.3 Hasil Analisis <i>Task</i> Aplikasi Kualitas Udara: Transportasi dan Poin	59
Tabel 4.4 Kebutuhan Fungsional	62
Tabel 4.5 Pertanyaan Kuesioner Kano.....	65
Tabel 4.6 Perhitungan Kuesioner Kano	66
Tabel 4.7 Hasil Pengujian <i>Unit Test</i> Servis Pengguna	90
Tabel 4.8 Hasil Pengujian <i>Unit Test</i> Servis Kualitas Udara	90
Tabel 4.9 Hasil Pengujian <i>Unit Test</i> Servis Perjalanan	91
Tabel 4.10 Hasil Pengujian <i>Unit Test</i> Servis Kupon dan Mitra Usaha.....	92
Tabel 4.11 Hasil Pengujian <i>Integration Test</i> Servis-Servis Mahoni.....	93
Tabel 5.1 Daftar <i>Endpoint</i>	98
Tabel 5.2 Skenario Pengujian Performa	99
Tabel 5.3 RabbitMQ dan/atau Kafka	101
Tabel 5.4 Daftar Kafka <i>Topic</i> Aplikasi Mahoni beserta Konfigurasinya	104
Tabel 5.5 Hasil Evaluasi Performa	124
Tabel 5.6 <i>Memory</i> dan <i>CPU Usage</i> pada Skenario Pertama	126
Tabel 5.7 <i>Memory</i> dan <i>CPU Usage</i> pada Skenario Kedua.....	126
Tabel 5.8 <i>Memory</i> dan <i>CPU Usage</i> pada Skenario Ketiga.....	126
Tabel 6.1 Skenario Pengujian <i>Scalability</i> dan <i>Reliability</i>	132
Tabel 6.2 Hasil Pengujian <i>Scalability</i> dan <i>Reliability</i>	171
Tabel 7.1 Kriteria Kelulusan Skenario <i>Thin-thread</i>	177
Tabel 7.2 Hasil Skenario <i>End-to-End Testing</i>	178
Tabel 7.3 Kriteria Kelulusan Evaluasi Kualitatif.....	179

DAFTAR KODE

Kode 5.1 Konfigurasi WAL pada PostgreSQL.....	112
Kode 5.2 Perintah Mengubah <i>Role User</i> pada PostgreSQL	113
Kode 5.3 Konfigurasi <i>docker-compose</i> Debezium	113
Kode 5.4 Contoh Konfigurasi Debezium <i>Source Connector</i>	114
Kode 5.5 Konfigurasi <i>docker-compose</i> Kafka Connect.....	116
Kode 5.6 Contoh Konfigurasi <i>Cassandra Sink Connector</i>	116
Kode 5.7 Contoh Konfigurasi Google Big Query <i>Sink Connector</i>	117
Kode 5.8 Konfigurasi <i>application.yml</i> pada Servis <i>User</i>	118
Kode 5.9 Pengiriman Event Menggunakan KafkaTemplate.....	119
Kode 5.10 Perhitungan Partisi Manual Berdasarkan <i>userId</i>	119
Kode 5.11 Contoh Consumer yang <i>Subscribe</i> ke <i>user-point-topic</i>	120
Kode 5.12 Potongan Kode Konfigurasi KafkaStreams.....	120
Kode 5.13 Potongan Kode Pembuatan Topologi KTable.....	121
Kode 5.14 Potongan Kode Pengambilan Data dari <i>State Store</i>	122
Kode 5.15 <i>Dockerfile</i> untuk Membuat Docker <i>image</i>	122
Kode 5.16 <i>Script</i> untuk Membuat dan Mengunggah Docker <i>image</i>	123
Kode 5.17 Konfigurasi Servis Loyalitas	127
Kode 5.18 <i>Schema</i> pada <i>user-point-topic</i>	128
Kode 6.1 <i>Query Skema Cassandra</i>	148
Kode 6.2 <i>Windowing Air Quality Job</i>	151
Kode 6.3 <i>Enrichment Air Quality Job</i>	153
Kode 6.4 <i>Sink InfluxDB Air Quality Job</i>	154
Kode 6.5 <i>Sink InfluxDB Trip Job</i>	156
Kode 6.6 Docker-compose InfluxDB	158
Kode 6.7 Docker-compose Grafana.....	159
Kode 6.8 Konfigurasi Konektor Topik <i>air-quality-processed-topic</i> dengan BigQuery	160
Kode 6.9 <i>Integration Test trip-topic</i>	163
Kode 7.1 Contoh Tes pada Postman	177

DAFTAR LAMPIRAN

Lampiran 1: Tabel Daftar Ambang Polutan untuk Konversi AQI	193
Lampiran 2: Hasil Wawancara	194
Lampiran 3: Kuesioner Kano	202
Lampiran 4: Class Diagram Servis Pengguna	208
Lampiran 5: Class Diagram Servis Kualitas Udara.....	209
Lampiran 6: Class Diagram Servis Perjalanan	210
Lampiran 7: Class Diagram Servis Kupon dan Mitra Usaha	211
Lampiran 8: Daftar Seluruh Endpoint Mahoni.....	212
Lampiran 9: Schema <i>Domain Events</i> Mahoni	218
Lampiran 10: <i>Response Body</i> Hasil Pengujian <i>End-to-End Testing</i> Evaluasi Umum .	220
Lampiran 11: Ringkasan Hasil Pengujian <i>End-to-End Testing</i> Evaluasi Umum.....	222
Lampiran 12 : File <i>deployment</i> Flink di Kubernetes Engine.....	223
Lampiran 13: Kode Konfigurasi docker-compose Kafka Cluster	228
Lampiran 14: Contoh Kode Konfigurasi KTable pada Servis <i>Trip</i>	230
Lampiran 15: Tabel Perhitungan Evaluasi Kano.....	231

BAB 1

PENDAHULUAN

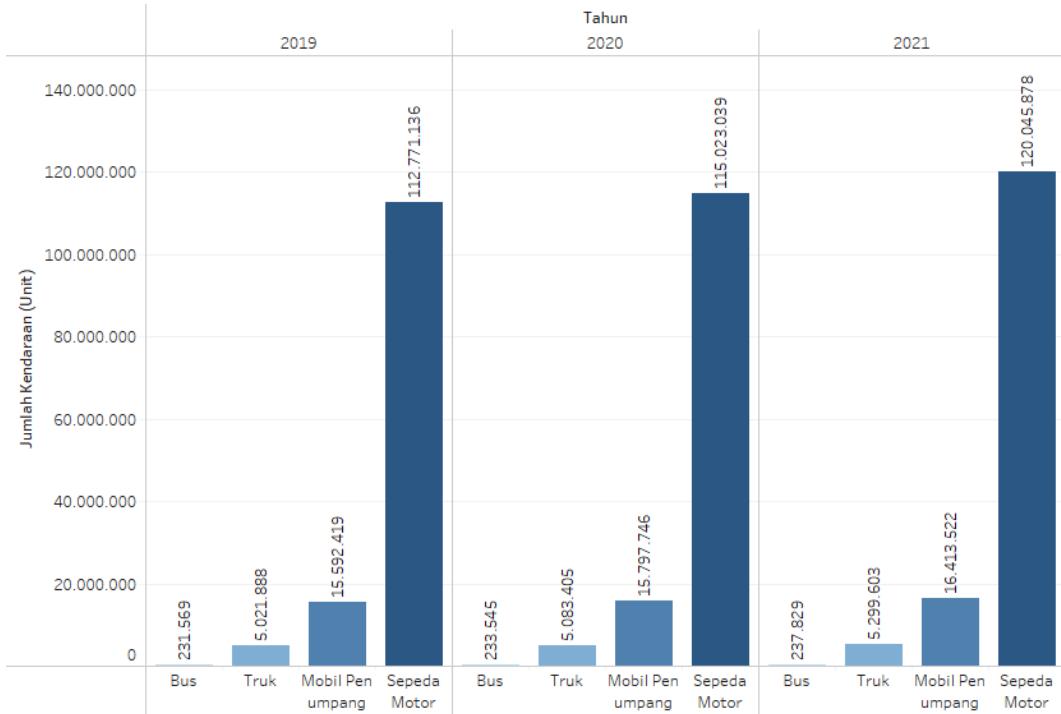
Bab ini berisi pendahuluan yang mencakup latar belakang pengembangan sistem aplikasi kota cerdas bernama Mahoni untuk mengatasi permasalahan kualitas udara yang buruk di perkotaan. Dalam bab ini juga mencakup pertanyaan penelitian, tujuan penelitian, batasan penelitian serta sistematika penulisan yang dijelaskan lebih lanjut dalam subbab.

1.1 Latar Belakang

Kota cerdas merupakan suatu konsep dalam kota modern yang menggunakan teknologi informasi dan komunikasi untuk meningkatkan kualitas hidup masyarakat dan juga pengelolaan daerah. Di Indonesia, terdapat program Gerakan Menuju 100 *Smart City* yang diinisiasi oleh berbagai instansi kementerian demi mengembangkan kota/kabupaten lebih lanjut dengan mengintegrasikan IT ke beberapa aspek dari kota cerdas salah satunya yaitu *smart mobility*. *Smart mobility* merupakan aspek transportasi dan aksesibilitas yang diintegrasikan dengan teknologi. Salah satu isu dari aspek transportasi di Indonesia adalah banyaknya jumlah kendaraan bermotor.

Perkembangan jumlah kendaraan bermotor di Indonesia selalu meningkat setiap tahunnya. Jumlah kendaraan bermotor pada tahun 2022 telah mencapai 152,51 juta unit, meningkat sebanyak 7,4% dari tahun sebelumnya (Sadya, 2023). Seperti data diolah dari Badan Pusat Statistik yang tertera dalam Gambar 1.1, apabila dilihat dari jenis kendaraannya, jumlah sepeda motor jauh lebih banyak dari jenis kendaraan lain seperti mobil penumpang, mobil bus, dan mobil barang.

Banyaknya kendaraan bermotor sangat memengaruhi tingkat kualitas udara di Indonesia (Mutiara, 2021). Sektor transportasi menyumbang peranan besar dalam pencemaran udara. (Ismiyati et al., 2014). Pencemaran udara dapat ditandai dengan tingginya tingkat polutan yang terkandung di udara. Polusi udara memiliki dampak buruk ke berbagai hal, baik ke lingkungan itu sendiri hingga kepada kesehatan makhluk hidup yang berada di sekitar area yang tercemar.



Gambar 1.1 Perkembangan Jumlah Kendaraan Bermotor menurut Jenis (Unit) Tahun 2019-2021

Sumber: Jumlah Kendaraan Bermotor Menurut Provinsi dan Jenis Kendaraan (unit), Badan Pusat Statistik, telah diolah kembali

Salah satu faktor utama seseorang untuk mengontrol dan mencegah polusi udara adalah kesadaran individu dan pengetahuannya akan tingkat kualitas udara (Liao et al., 2019). Untuk mengurangi polusi udara karena sektor transportasi, pendekatan yang dapat dilakukan adalah peralihan penggunaan kendaraan pribadi ke kendaraan umum secara massal. Untuk meningkatkan kesadaran masyarakat mengenai tingkat kualitas udara, telah banyak dikembangkan aplikasi yang dapat memonitor kualitas udara. Namun dalam studi literatur dan observasi yang telah dilakukan, belum ada aplikasi yang mendukung peningkatan penggunaan transportasi umum serta memonitor kualitas udara secara bersamaan.

Oleh karena itu, dikembangkan suatu sistem aplikasi kota pintar bernama Mahoni yang menghubungkan beberapa aspek kota cerdas yaitu aspek *smart environment*, *smart mobility*, dan *smart economy*. Mahoni dapat menginformasikan kepada pengguna mengenai kualitas udara di sekitarnya (aspek *smart environment*) serta mendukung penggunaan transportasi umum dengan insentif pemberian poin yang kuantitasnya relatif

terhadap tingkat kualitas udara (aspek *smart mobility*). Poin yang telah dikumpulkan dapat ditukarkan menjadi kupon potongan harga, tiket masuk museum, ataupun hadiah lainnya yang dapat disesuaikan dengan perkembangan perekonomian di daerah tersebut (aspek *smart economy*).

Mahoni merupakan konsep aplikasi kota cerdas hasil lomba GEMASTIK XIV 2021 kelompok TaPir yang terdiri dari Riswanda, Natasya, dan Adamy. Luaran yang telah dihasilkan kelompok TaPir dari lomba tersebut adalah konsep aplikasi Mahoni dan purwarupa fidelitas tinggi saja. Pada luaran tersebut, belum terdapat adanya validasi fitur dari pengguna sehingga pada penelitian ini validasi fitur dilakukan untuk menentukan apakah fitur-fitur yang dapat dibuat sesuai dengan kebutuhan pengguna. Selain itu, implementasi untuk Mahoni juga belum dilakukan sehingga pada penelitian ini arsitektur dari layanan aplikasi Mahoni dirancang dan diimplementasikan berdasarkan hasil dari validasi fitur yang dilakukan.

Mahoni mengolah data-data kualitas udara yang didapatkan melalui sensor-sensor udara serta data-data penggunaan aplikasi. Data-data tersebut bersifat kompleks dan besar sehingga membutuhkan arsitektur *big data* dalam aplikasi Mahoni. Big data merupakan kumpulan data yang sangat kompleks yang memiliki sifat 3V yaitu *volume*, *velocity* dan *variety* (Balusamy et al., 2021). *Velocity* merupakan faktor yang sulit diatasi karena harus menangani data secara *real-time* setiap detiknya (Feick et al., 2018). Hal tersebut mendukung penggunaan arsitektur *big data* untuk kota cerdas dan memiliki peranan penting di dalam implementasi kota cerdas (Hashem et al., 2016). Arsitektur *big data* ini dibuat untuk mengolah data secara *real-time* serta dapat menampilkan dan menyimpan analisis melalui sebuah *dashboard* secara *real-time*.

Selain merancang dan mengimplementasikan arsitektur *big data*, diperlukan juga arsitektur untuk mengatur komunikasi antar servis melalui sebuah *event*. Mengambil sudut pandang kota cerdas, konteks *event* merepresentasikan lebih dari sekadar perubahan data. Sebuah *event* adalah informasi penting atas sesuatu yang terjadi di dalam lingkup kota cerdas tersebut baik itu sensor, transportasi umum, maupun aktivitas manusia (Cretu, 2012). Oleh karena itu, sebuah aplikasi kota cerdas harus bisa menghubungkan banyak komponen yang saling berkomunikasi satu sama lain. Kebutuhan ini menciptakan

ketergantungan antar sistem yang kemudian dapat membuat sistem konvensional menjadi *tightly-coupled* sehingga sulit untuk dibuat dan dikembangkan dalam skala yang besar.

Arsitektur *event-driven* dapat melepas ketergantungan antar sistem dengan menggunakan *event* dan *message broker* sebagai mediator untuk komunikasi (Stopford, 2018). Sebuah *event* dapat memicu reaksi pada komponen untuk melakukan sesuatu, hal ini menyebabkan komunikasi berjalan secara *asynchronous* dan mengurangi *coupling* antar komponen pada arsitektur. Salah satu *message broker* yang dapat digunakan di dalam arsitektur *event-driven* adalah Kafka. Kafka merupakan sebuah *message broker* yang paling populer digunakan untuk kebutuhan memproses big data stream karena sifatnya *scalable* dan memiliki *throughput* tinggi (Hiraman et al, 2018). Terlebih lagi, terdapat sebuah riset yang dilakukan oleh Winberg et al pada 2021 yang menyimpulkan bahwa arsitektur *event-driven* dapat digunakan untuk membangun aplikasi kualitas udara *real-time* dengan *throughput* yang cukup baik. Oleh karena itu, pada penelitian ini juga diusulkan untuk menggunakan arsitektur *event-driven* dengan Kafka dalam mengembangkan aplikasi kota cerdas Mahoni.

1.2 Pertanyaan Penelitian

Berdasarkan latar belakang tersebut, terdapat beberapa pertanyaan penelitian yang ingin dijawab pada penelitian ini yaitu:

1. Apa fitur-fitur yang dapat dibuat dan dikembangkan untuk aplikasi Mahoni?
 - a. Bagaimana merancang fitur yang sesuai dengan kebutuhan pengguna untuk aplikasi Mahoni?
 - b. Bagaimana mengimplementasikan servis-servis berdasarkan fitur yang dapat dibuat?
2. Bagaimana merancang arsitektur *big data* untuk pengolahan data aplikasi Mahoni?
 - a. Bagaimana mengimplementasi *layer-layer* yang ada pada arsitektur dengan menggunakan *framework* yang tepat?

- b. Bagaimana mengimplementasi *output* dalam bentuk *dashboard* yang akurat?
 - c. Bagaimana membuat arsitektur *big data* yang *scalable* dan *reliable*?
3. Bagaimana membuat arsitektur *event-driven* untuk aplikasi Mahoni ?
 - a. Bagaimana membangun arsitektur yang *loosely-coupled*?
 - b. Bagaimana membangun arsitektur *event-driven* dengan *throughput* yang tinggi?

1.3 Tujuan Penelitian

Berdasarkan pertanyaan penelitian yang telah dipaparkan, tujuan yang ingin dicapai dari penelitian ini yaitu:

1. Merancang dan mengimplementasikan layanan fitur-fitur yang sesuai dengan kebutuhan pengguna untuk menggunakan aplikasi Mahoni.
2. Merancang, mengimplementasikan, dan mengevaluasi arsitektur *big data* untuk aplikasi Mahoni.
3. Merancang, mengimplementasikan, dan mengevaluasi arsitektur *event-driven* untuk aplikasi Mahoni.

1.4 Batasan Penelitian

Di dalam penelitian ini, terdapat beberapa batasan ruang lingkup pekerjaan yang dikerjakan yaitu:

1. Pengembangan aplikasi ini hanya dibatasi pada bagian *backend* yang terdiri dari *codebase* servis, pengolahan dan analisis *big data*, dan *message broker*.
2. Penelitian ini tidak mencakup desain UI/UX, pengembangan *frontend*, dan pengembangan arsitektur untuk perangkat IoT untuk sumber data kualitas udara.
3. Data yang digunakan merupakan data *dummy* dan duplikasi yang diolah berdasarkan data asli.

1.5 Sistematika Penulisan

Di dalam penelitian ini, terdapat beberapa batasan ruang lingkup pekerjaan yang dikerjakan yaitu:

1. Bab 1 Pendahuluan, membahas latar belakang penelitian diadakan termasuk pertanyaan penelitian, tujuan serta batasan-batasan yang dikerjakan pada penelitian ini.
2. Bab 2 Studi Literatur, membahas penjelasan mengenai konsep-konsep yang digunakan serta memberikan penjelasan pada terminologi yang digunakan.
3. Bab 3 Metode dan Metodologi Umum, membahas metode-metode yang digunakan untuk menyelesaikan masalah yang diangkat pada penelitian ini dan metodologi yang dilakukan secara umum.
4. Bab 4 Pengembangan Servis-Servis Mahoni, membahas lebih detail mengenai metodologi, praperancangan, perancangan, pengembangan, dan evaluasi servis-servis Mahoni.
5. Bab 5 Arsitektur *Event-Driven* Mahoni, membahas lebih detail mengenai metodologi, perancangan, pengimplementasian, dan evaluasi hasil arsitektur *event-driven*.
6. Bab 6 Arsitektur *Big Data* Mahoni, membahas lebih detail mengenai metodologi, praperancangan, perancangan, pengimplementasian arsitektur, dan evaluasi hasil arsitektur *big data*.
7. Bab 7 Evaluasi dan Kesimpulan Umum, membahas evaluasi yang ditujukan untuk keseluruhan sistem secara umum serta kesimpulan, kritik, dan saran yang dapat dilakukan untuk pengembangan aplikasi Mahoni ke depan.

BAB 2

TINJAUAN PUSTAKA

Bab ini memaparkan tinjauan pustaka yang menunjang penelitian ini. Tinjauan pustaka tersebut menjelaskan konsep yang digunakan dalam aplikasi Mahoni, terminologi metodologi, arsitektur sistem dan data, dan penyimpanan data. Selain itu, terdapat juga pembahasan konsep terkait *stream processing*, Apache Kafka, dan *change data capture*.

2.1 Konsep Aplikasi

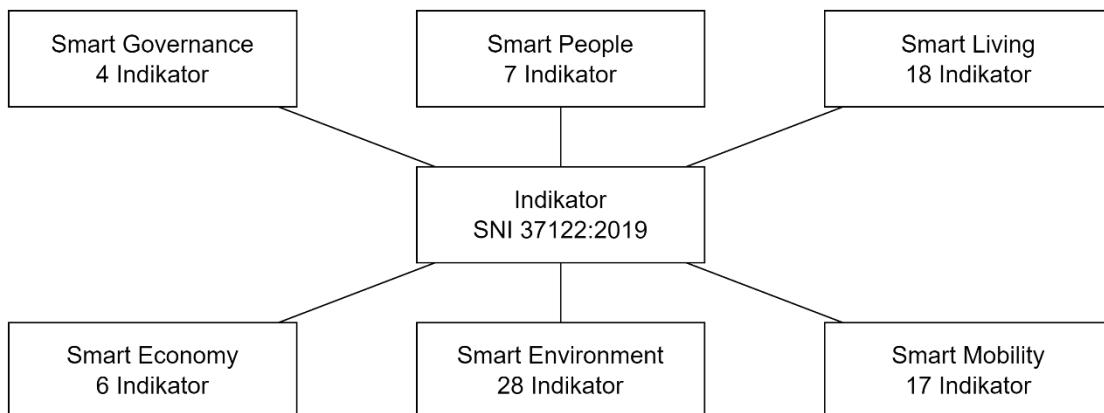
Aplikasi Mahoni memiliki beberapa tema yang menjadi identitas dari aplikasi itu sendiri. Pada subbab ini, dijelaskan konsep yang berkaitan dengan tema tersebut yaitu kota cerdas, indeks kualitas udara, dan sistem penghargaan poin. Pembahasan terakhir tidak hanya terkait dengan definisi, tetapi juga kepada bagaimana konsep tersebut diterapkan oleh Mahoni.

2.1.1 Kota Cerdas

Pengembangan aplikasi Mahoni terinspirasi dari implementasi aplikasi sebagai upaya solusi permasalahan kota yang telah dibahas di subbab latar belakang pada bab 1 dengan pendekatan kota cerdas. Istilah kota cerdas semakin popular seiring gencarnya perkembangan pembangunan dan teknologi yang terintegrasi dengan suatu kota atau daerah. Saat ini, terdapat banyak definisi yang menjelaskan apa itu kota cerdas seperti pada tinjauan literatur yang dilakukan oleh Winkowska et al. (2019). Secara umum, istilah kota cerdas digunakan untuk merujuk kota yang memanfaatkan teknologi dan pengelolaan informasi di berbagai aspek dalam kota tersebut demi mengusung kehidupan penduduk yang lebih baik. Terdapat acuan suatu kota dapat dikatakan kota cerdas, yaitu apabila memenuhi indikator yang tertera di SNI 37122:2019 yang merupakan translasi dari ISO 37122. Seperti yang dapat dilihat pada Gambar 2.1, indikator tersebut mencakup enam pilar kota cerdas, yaitu *smart governance*, *smart people*, *smart living*, *smart mobility*, *smart environment*, dan *smart economy* (Direktorat Jenderal Aplikasi Informatika Kementerian Komunikasi dan Informatika, 2021).

Peningkatan layanan publik, kolaborasi antara masyarakat dengan pemerintah, dan transparansi serta efisiensi dalam pemerintahan yang melibatkan penggunaan TIK adalah

tujuan dari *smart governance*. Tingkat akses terhadap informasi serta pemberdayaan, keterbukaan, dan keterlibatan masyarakat dalam komunitas melalui teknologi merupakan tolak ukur dari *smart people*. Peningkatan kualitas hidup dan sosial melalui penyediaan fasilitas sosial menjadi indikator dari *smart living*. Pengembangan transportasi berkelanjutan dan peningkatan mobilitas dengan integrasi teknologi merupakan fokus dari *smart mobility*. Penggunaan teknologi untuk melindungi dan meningkatkan lingkungan alam serta sumber daya, pengelolaan sampah, efisiensi energi, dan mengatasi polusi adalah inti dari *smart environment*. Kolaborasi setiap pihak (masyarakat, bisnis, dan pemerintah) serta pertumbuhan ekonomi dan inovasi kewirausahaan melalui teknologi menjadi tolak ukur dari *smart economy* (Winkowska et al., 2019, dalam Stawasz & Sikora-Fernandez, 2016; Zanella et al., 2014; Caragliu et al., 2011).



Gambar 2.1 SNI 37122:2019 Sebagai Indikator Kota Cerdas

Sumber: Direktorat Jenderal Aplikasi Informatika Kementerian Komunikasi dan Informatika (2021), telah diolah kembali

Mahoni mengupayakan implementasi aplikasi yang berorientasi pada tiga pilar kota cerdas, yaitu *smart environment*, *smart mobility*, dan *smart economy*. Penerapan aspek *smart environment* dicapai dengan implementasi pemantauan kualitas udara, penerapan aspek *smart mobility* dicapai dengan implementasi sistem penghargaan poin ketika menaiki transportasi umum, sedangkan penerapan aspek *smart economy* dicapai dengan implementasi sistem tukar poin menjadi kupon. Poin Mahoni menjadi penengah dari pilar atau aspek tersebut, yang mana besarnya dipengaruhi oleh kualitas udara, didapatkan dengan menaiki transportasi umum, dan ditukarkan dengan kupon yang tersedia.

2.1.2 Indeks Kualitas Udara

Salah satu dari tiga sistem yang diterapkan Mahoni adalah pemantau kualitas udara. Udara merupakan salah satu komponen terpenting untuk kehidupan di muka bumi. Di antara unsur atau senyawa yang terkandung di udara seperti oksigen, nitrogen, helium, dan karbon dioksida, ada yang termasuk ke dalam golongan polutan udara. Polutan udara merupakan unsur atau senyawa yang apabila terkandung dalam jumlah yang besar maka dapat menyebabkan polusi udara. Untuk mengukur tingkat polusi udara dengan menghitung kadar polutan, digunakan satuan yang umumnya adalah *air quality index*.

Air quality index (AQI) atau indeks kualitas udara merepresentasikan kondisi kualitas udara dalam suatu area terbuka. Terdapat beraneka ragam model perhitungan indeks kualitas udara yang tersedia untuk mengukur tingkat kualitas udara. Keunikan geografis, perbedaan kebutuhan dan kondisi udara pada suatu daerah dengan daerah lainnya, serta perkembangan dari sensor udara dari segi kualitas dan harga seiring berjalannya waktu menjadi pemicu dikembangkannya model perhitungan indeks kualitas udara yang berbeda (Kumar, 2022). Kendati dari banyaknya jenis model perhitungan indeks kualitas udara, terdapat salah satu pendekatan yang umumnya digunakan oleh aplikasi pemantau kualitas udara, yaitu perhitungan indeks kualitas udara yang diusung oleh Air Quality Assessment Division (2018) dari badan United States Environmental Protection Agency.

Kalkulasi indeks kualitas udara menurut Air Quality Assessment Division (2018) adalah dengan mempertimbangkan konsentrasi ozon, PM2.5, PM10, CO, NO2, dan SO2 yang terkandung di udara pada suatu daerah terbuka dalam suatu rentang waktu. Perhitungan dapat dilakukan secara *real-time* setiap satu jam sekali, secara agregat seperti delapan jam sekali, atau 24 jam sekali. Kemudian, nilai dari indeks kualitas udara diambil berdasarkan hasil perhitungan tertinggi dari tiap polutannya. Formula dari perhitungan indeks kualitas udara untuk setiap polutannya dapat dilihat pada Persamaan 2.1. Daftar ambang konsentrasi polutan yang terkandung untuk kalkulasi formula terdapat pada Lampiran 1.

$$I_p = \frac{I_{Hi} - I_{Lo}}{BP_{Hi} - BP_{Lo}} (C_p - BP_{Lo}) + I_{Lo} \quad (2.1)$$

Nilai dari indeks kualitas udara yang diperoleh dapat diklasifikasikan menjadi beberapa kategori yang menunjukkan kualitas udara dari suatu daerah pada waktu tertentu. Penentuan hal tersebut dapat dilakukan dengan mengacu kepada Tabel 2.1. Terdapat enam kategori pengelompokan tingkat kualitas udara, yaitu *good*, *moderate*, *unhealthy for sensitive group*, *unhealthy*, *very unhealthy*, dan *hazardous*.

Tabel 2.1 Kategori Tingkat Kualitas Udara berdasarkan AQI

Kategori	AQI	Warna Representasi
<i>Good</i>	0–50	Hijau
<i>Moderate</i>	51–100	Kuning
<i>Unhealthy for sensitive group</i>	100–150	Jingga
<i>Unhealthy</i>	151–200	Merah
<i>Very unhealthy</i>	201–300	Ungu
<i>Hazardous</i>	Lebih dari 300	Marun

Kualitas udara dapat mempengaruhi manusia dalam melakukan aktivitas atau kegiatan. Maka dari itu, indeks kualitas udara beserta dengan kategorinya bisa menjadi acuan untuk berkegiatan khususnya apabila dilakukan di luar ruangan. Kategori kualitas udara di mana udara tersebut baik untuk manusia adalah *good*. Kualitas udara dengan kategori *moderate* masih aman untuk manusia. Dimulai dari kategori *unhealthy for sensitive group*, kualitas udara dikatakan buruk dan mempengaruhi kesehatan.

Salah satu tindakan pencegahan terhadap dampak negatif yang ditimbulkan dari kualitas udara yang buruk adalah dengan memberi peringatan atau saran mengenai hal-hal untuk dilakukan ataupun dihindari. Pencegahan tersebut dapat diimplementasikan pada aplikasi pemantau kualitas udara sehingga tidak hanya memberikan informasi mengenai kualitas udara, tetapi juga interpretasi dan saran tindakan mengenai hal tersebut.

2.1.3 Sistem Penghargaan Poin

Selain pemantau kualitas udara, Mahoni menerapkan sistem penghargaan poin yang diterapkan pada fitur perjalanan transportasi umum dan penukaran poin menjadi kupon. Poin merupakan satuan nilai yang dapat digunakan sebagai bentuk penghargaan virtual (*virtual reward*) kepada pengguna. Penghargaan diberikan sebagai motivasi ataupun pereda kekecewaan atas kerja keras yang dilakukan pengguna (Wang dan Sun, 2011).

Setiap bentuk penghargaan dalam sistem penghargaan memiliki kegunaan dan tujuannya masing-masing. Dalam klasifikasi bentuk penghargaan yang dilakukan oleh Wang dan Sun (2011), poin yang digunakan sebagai mata uang seperti penggunaan poin dalam sistem Mahoni dapat termasuk kepada dua bentuk penghargaan yaitu skor dan barang berharga. Bentuk penghargaan skor menggambarkan tolak ukur atas pekerjaan yang dilakukan untuk perhitungan nilainya dan umumnya digunakan sebagai perbandingan. Namun, dalam hal ini poin tersebut juga tergolong sebagai barang berharga karena digunakan secara praktis sebagai alat tukar. Penerapan sistem penghargaan poin dengan tujuan tersebut dapat memotivasi pengguna untuk mengumpulkan poin sebanyak-banyaknya agar nantinya dipergunakan dalam transaksi.

2.1.3.1 Implementasi Terkait

Seperti yang telah disebutkan sebelumnya pada subbab 2.1.1, Mahoni memanfaatkan kualitas udara dalam kalkulasi poin. Tidak hanya Mahoni yang memiliki sistem yang memberikan poin dengan konsiderasi lingkungan dalam perhitungannya, telah terdapat implementasi skema insentif penggunaan aplikasi lingkungan berbasis *mobile* pada pengembangan aplikasi oleh Huang et al. (2022). Aplikasi tersebut menggunakan poin sebagai bentuk penghargaan kepada penggunanya dengan sistem penghargaan yang mirip dengan Mahoni tetapi memiliki perbedaan pada proses bisnisnya.

Huang et al. (2022) melakukan pengembangan aplikasi daur ulang limbah elektronik yang menerapkan sistem penghargaan poin. Aplikasi ini menerapkan skema insentif ekologis di mana pemberian poin bertujuan untuk memotivasi pengguna melakukan daur ulang. Poin dengan sebutan *eco-credits* didapatkan setelah mendaur ulang limbah elektronik di sebuah *end-device* tempat sampah pintar. Besarnya *eco-credits* ditentukan dari jenis limbah yang didaur ulang. *Eco-credits* yang dikumpulkan dapat ditukarkan dengan berbagai macam hal seperti diskon, tiket, ataupun didonasikan. Skema tersebut mempunyai andil dalam konsep ekonomi sirkular yang menjadi latar belakangnya.

2.1.3.2 Penelitian Terkait

Pemberian insentif ataupun penghargaan kepada pengguna ketika telah menyelesaikan suatu pekerjaan tertentu untuk memotivasi pengguna dalam menggunakan aplikasi telah banyak diterapkan di berbagai bidang. Penelitian mengenai penerapan sistem tersebut di

antaranya ditemukan pada aplikasi sosial oleh Farzan et al. (2008), bisnis oleh Hwang dan Choi (2019), permainan oleh Goh et al. (2017), serta lingkungan oleh Zhong dan Huang (2016). Evaluasi dan hasil dari penelitian implementasi sistem penghargaan berupa poin di berbagai bidang menjadi masukan untuk pengembangan sistem penghargaan poin untuk sistem penukaran poin menjadi kupon Mahoni. Tentunya, implementasi sistem penghargaan poin Mahoni menyesuaikan proses bisnis dan kebutuhan dari sistem sehingga terdapat persamaan maupun perbedaan dengan implementasi pada penelitian terdahulu seperti yang tertera pada Tabel 2.2.

Tabel 2.2 Penelitian Terdahulu Implementasi Sistem Penghargaan Poin

No.	Judul	Penulis (Tahun)	Hasil Penelitian	Persamaan	Perbedaan
1.	<i>Results from Deploying a Participation Incentive Mechanism Within the Enterprise</i>	Farzan et al. (2008)	Sistem poin memotivasi pengguna untuk berkontribusi lebih.	Pemberian nilai poin yang dinamis.	Tidak menerapkan mekanika penggunaan poin.
2.	<i>Having Fun while Receiving Rewards?: Exploration of Gamification in Loyalty Programs for Consumer Loyalty</i>	Hwang dan Choi (2019)	Gamifikasi meningkatkan efektifitas dan pengalaman dalam meningkatkan loyalitas pengguna.	Melakukan gamifikasi proses bisnis.	Menggunakan kombinasi bentuk penghargaan.
3.	<i>Perceptions of Virtual Reward Systems in Crowdsourcing Games.</i>	Goh et al. (2017)	Pemberian penghargaan meningkatkan kesenangan dan motivasi.	Pemberian nilai poin yang dinamis.	Menggunakan kombinasi bentuk penghargaan.
4.	<i>The Empirical Research on the Consumers' Willingness to Participate in E-waste Recycling with a Points Reward System</i>	Zhong dan Huang (2016)	Sistem poin memotivasi pengguna dan membantu sistem manajemen.	Melakukan gamifikasi dan proses bisnis.	Implementasi sistem proses bisnis.

Farzan et al. (2008) mengevaluasi penerapan sistem insentif dalam situs aplikasi sosial antar pekerja di perusahaan. Tujuan dari penerapan sistem insentif tersebut adalah untuk meningkatkan kontribusi pengguna dalam situs aplikasi. Bentuk insentif yang digunakan adalah poin dan status pengguna. Poin didapatkan pengguna setelah memberikan komentar, membuat konten, dan melengkapi profil. Kemudian, pengguna mendapatkan status yang merepresentasikan banyaknya poin yang dikumpulkannya. Evaluasi yang didapatkan dari penerapan sistem insentif tersebut adalah sistem insentif berhasil membuat pengguna berkontribusi lebih karena termotivasi dalam mengumpulkan poin dan meningkatkan statusnya. Namun, efek yang didapatkan tidak bertahan lama karena pengguna cenderung berhenti ketika merasa poin dan status yang dimilikinya sudah cukup. Oleh karena itu, dalam evaluasi ini disarankan untuk menciptakan sistem insentif yang dinamis seiring berjalannya waktu sehingga cukup untuk membuat pengguna terus termotivasi menggunakan sistem insentif tersebut.

Hwang dan Choi (2019) menginvestigasi pengaruh gamifikasi pemberian poin pada program loyalitas dalam bisnis. Dalam studi kasus di penelitian tersebut, poin didapatkan pengguna setelah menyelesaikan tugas atau tantangan yang terdapat pada *bingo* sebagai bentuk gamifikasi yang dilakukan. Poin yang telah terakumulasi dapat ditukarkan dalam berbagai bentuk hadiah yang ditawarkan program loyalitas. Penelitian ini menunjukkan bahwa program loyalitas dengan gamifikasi lebih efektif dan menawarkan lebih banyak pengalaman dalam meningkatkan loyalitas pengguna dibandingkan dengan program loyalitas konvensional. Hasil dari penelitian ini mendukung potensi dari gamifikasi yang dapat menunjang hubungan manajemen bisnis terhadap pelanggannya.

Goh et al. (2017) menginvestigasi pengaruh penerapan sistem penghargaan virtual di aplikasi permainan *crowdsourcing* yang menggunakan pemberian poin dan pengumpulan lencana. Dalam aplikasi tersebut, poin dan lencana diberikan atas aksi yang dilakukan oleh pengguna seperti kontribusi pembuatan kiriman (*post*), pemberian reaksi dan penilaian terhadap kiriman, dan sebagainya. Dari hasil dari penelitian yang dilakukan, ditemukan bahwa pemberian penghargaan meningkatkan kesenangan secara emosional, kognitif, dan perilaku, serta persepsi atas kualitas luaran dari aktivitas yang dilakukan. Selain itu, pemberian penghargaan memenuhi kebutuhan motivasi pengguna terkait kebebasan bertindak, penentuan pilihan, dan penghadapan tantangan dalam aplikasi.

Zhong dan Huang (2016) meneliti kesediaan pengguna berpartisipasi dalam sistem daur ulang sampah elektronik yang menerapkan sistem penghargaan poin. Penelitian ini menjadi dasar pengembangan aplikasi daur ulang oleh Huang et al. (2022). Dalam penelitian tersebut, ditemukan bahwa terdapat korelasi positif antara kesediaan partisipasi pengguna dalam penggunaan sistem daur ulang yang menerapkan sistem penghargaan poin dengan ekspektasi, pertimbangan nilai, dan kesediaan pengguna dalam penggunaan sistem daur ulang konvensional. Hasil analisis menunjukkan bahwa sistem penghargaan poin tersebut memotivasi pengguna dan membantu sistem manajemen daur ulang.

Berdasarkan penerapan sistem penghargaan poin pada beberapa paparan penelitian di atas, disimpulkan bahwa sistem penghargaan poin dapat memotivasi pengguna dalam melakukan suatu hal pada sistem. Pada Mahoni, sistem penghargaan poin ditujukan untuk memotivasi pengguna dalam menggunakan transportasi umum dan mencatatnya dalam aplikasi Mahoni. Kemudian untuk menjaga motivasi tersebut, poin yang telah dikumpulkan dapat ditukarkan menjadi kupon yang nantinya dapat dipakai oleh pengguna.

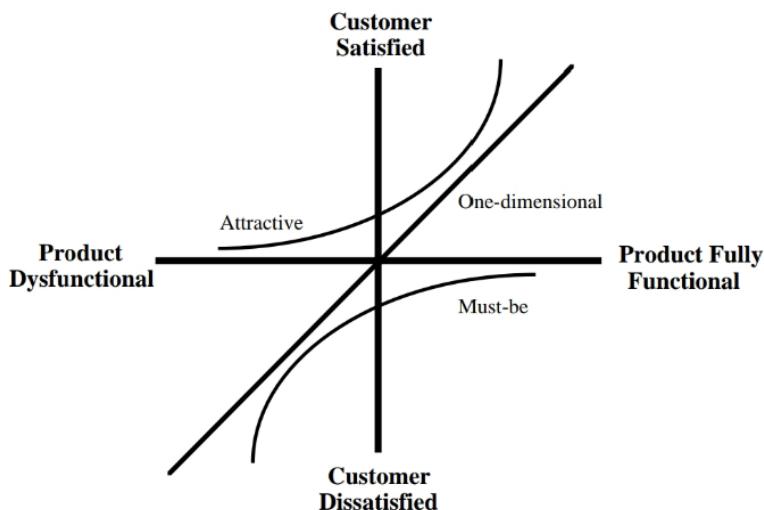
2.2 Terminologi Metodologi

Selama melakukan pekerjaan, terdapat beberapa terminologi yang digunakan yaitu Kano *method* dan Scrumban. Kano *method* merupakan metode yang digunakan dalam tahap *requirement gathering* di mana pembahasan lengkapnya terdapat pada subbab 4.2.8. Kemudian, Scrumban merupakan *framework* yang digunakan selama pengembangan baik dalam pekerjaan bab 4, bab 5, maupun bab 6.

2.2.1 Kano *Method*

Model Kano atau Kano *method* merupakan seperangkat konsep oleh Kano (1984) yang menjelaskan hubungan kepuasan pengguna dengan fungsionalitas aspek-aspek tertentu yang ditawarkan dari produk atau jasa. Hubungan kepuasan dengan fungsionalitas tersebut dapat digambarkan dengan diagram dua dimensi seperti pada Gambar 2.2 (Berger et al., 1993). Dari pemetaan hubungan tersebut, suatu aspek yang ditawarkan dari produk atau jasa dapat dikategorikan menjadi empat, yaitu:

1. *Attractive*, yaitu apabila produk atau jasa menghadirkan aspek fungsionalitas tertentu, maka kepuasan pengguna semakin meningkat. Namun, apabila produk atau jasa kurang atau tidak menghadirkan aspek fungsionalitas tertentu, maka kepuasan pengguna tidak begitu terpengaruh.
2. *Must-be*, yaitu apabila produk atau jasa menghadirkan aspek fungsionalitas tertentu, maka kepuasan pengguna tidak begitu terpengaruh. Namun, apabila produk atau jasa kurang atau tidak menghadirkan aspek fungsionalitas tertentu, maka kepuasan pengguna semakin menurun.
3. *One-dimensional*, yaitu produk atau jasa menghadirkan aspek fungsionalitas tertentu, maka kepuasan pengguna semakin meningkat. Namun, apabila produk atau jasa kurang atau tidak menghadirkan aspek fungsionalitas tertentu, maka kepuasan pengguna semakin menurun.
4. *Indifferent*, yaitu kepuasan pengguna tidak begitu terpengaruh apabila produk atau jasa menghadirkan aspek fungsionalitas tertentu ataupun tidak.



Gambar 2.2 Diagram Kano

Sumber: Berger et al. (1993)

Penerapan dari Kano *method* sering digunakan dalam manajemen mutu untuk menentukan dan memprioritaskan fitur dari suatu produk atau jasa yang paling penting

bagi pengguna. Dalam penelitian ini, Kano *method* digunakan untuk memprioritaskan fitur Mahoni. Pengumpulan data untuk melakukan analisis aspek dari suatu fitur dengan Kano *method* dapat dilakukan menggunakan kuesioner. Untuk setiap aspek dari produk atau jasa, pada kuesioner dituliskan pertanyaan yang menanyakan bagaimana pendapat responden mengenai fungsionalitas dan disfungsionalitas aspek tersebut. Jawaban dari pertanyaan tersebut adalah berupa pilihan-pilihan yang sama untuk setiap pertanyaannya.

Aspek yang ditawarkan dari produk atau jasa kemudian dikategorikan menjadi enam berdasarkan jawaban dari kuesioner, yaitu empat kategori yang sebelumnya disebutkan dan dua aspek lain yaitu *reverse* dan *questionable*. *Reverse* yaitu pemahaman responden kuesioner mengenai pertanyaan fungsionalitas dan disfungsionalitas terbalik. *Questionable* yaitu apabila jawaban pertanyaan fungsionalitas dan disfungsionalitas responden tidak konsisten. Klasifikasi kategori jawaban responden dilakukan dengan menggunakan tabel evaluasi Kano seperti pada Tabel 2.3 (Berger et al., 1993). Kemudian, kategori aspek yang ditawarkan produk atau jasa didapatkan dengan menghitung modus dari kategori aspek tersebut. Urutan prioritas aspek produk atau jasa berdasarkan kategori yang diperoleh adalah sebagai berikut: *Must-be* > *One-dimensional* > *Attractive* > *Indifferent*.

Tabel 2.3 Tabel Evaluasi Kano

Customer Requirement		Dysfunctional				
		1. like	2. must-be	3. neutral	4. live with	5. dislike
Functional	1. like	Q	A	A	A	O
	2. must-be	R	I	I	I	M
	3. neutral	R	I	I	I	M
	4. live-with	R	I	I	I	M
	5. dislike	R	R	R	R	Q

Keterangan: Attractive (A), Must-be (M), Reverse (R), One-dimensional (O), Questionable result (Q), Indifferent (I). Sumber: Berger et al. (1993)

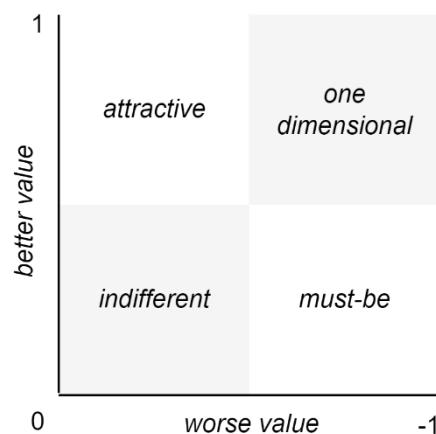
Kategori dari aspek suatu produk atau jasa yang telah diketahui setelah mengolah data kuesioner kemudian menjadi dasar untuk menetapkan prioritas kepentingan aspek tersebut. dan dapat menjadi acuan untuk tim pengembang produk atau jasa dalam melakukan kegiatan produksi. Namun, terkadang banyak aspek yang memiliki kategori

yang sama sehingga sulit untuk menentukan prioritas dari aspek tersebut. Oleh karena itu, analisis lanjutan dapat dilakukan dengan menghitung skor positif (*Better value*) dan negatif (*Worse value*) (Berger, 1993). Rumus perhitungan skor positif dan negatif dapat dilihat pada Persamaan 2.2 dan Persamaan 2.3.

$$Better = \frac{A + O}{A + O + M + I} \quad (2.2)$$

$$Worse = -\frac{O + M}{A + O + M + I} \quad (2.3)$$

Skor untuk setiap aspek yang telah dihitung dipetakan ke dalam diagram dua dimensi seperti pada Gambar 2.3 (Berger et al., 1993). Diagram tersebut menggambarkan persebaran posisi dari setiap aspek berdasarkan kepentingannya. Selain itu, informasi mengenai kategori dari suatu aspek masih dapat terlihat yaitu di kuadran mana aspek tersebut berada.



Gambar 2.3 Diagram Representasi Kategori Kano

Sumber: Berger et al. (1993), telah diolah kembali

Prioritisasi fitur dalam pekerjaan ini didasarkan hasil yang tertera pada hasil diagram representasi kategori Kano. Berdasarkan urutan prioritas aspek yang telah dijelaskan sebelumnya, maka apabila dilihat dari diagram tersebut, prioritas aspek yang dipilih dimulai secara berlawanan arah jarum jam dari kuadran di pojok kanan bawah diagram yaitu aspek *must-be*, kemudian aspek *one-dimensional*, *attractive*, dan *indifferent*.

2.2.2 Scrumban

Agar proses manajemen pekerjaan lebih terkelola, dibutuhkan penggunaan suatu kaidah yang mengatur bagaimana cara bekerja. Pemilihan kaidah tersebut tergantung bagaimana proses pekerjaan yang dilakukan. Proses pengembangan dalam penelitian ini dilakukan secara adaptif dan berkelanjutan yang mana serupa dengan sifat dari metodologi pengembangan *agile*. Untuk menerapkan metodologi tersebut, Scrumban dipilih sebagai *framework* yang digunakan.

Scrumban merupakan perpaduan antara dua *framework* populer yang berbeda, yaitu Scrum dan Kanban. Scrumban hadir dengan menyediakan transparansi manajemen pekerjaan dengan visualisasi kondisi pekerjaan dengan Kanban *board* dan seperangkat praktik yang terstruktur dari Scrum (Bhavsar et al., 2020). Masing-masing *framework* memiliki karakteristik yang beberapa di antaranya membentuk karakteristik dari Scrumban. Peng gabungan karakteristik tersebut yang dilakukan oleh Bhavsar et al. (2020) adalah dengan menghilangkan kekakuan Scrum dan mengambil karakteristik Kanban yang dapat mengatasi kebutuhan yang tidak dimiliki dari Scrum. Scrumban mengambil ritual Sprint serta sistem *pull* dan *push* dari Scrum, sedangkan elemen pembatasan WIP dan CICD diambil dari Kanban. Karakteristik yang direpresentasikan sebagai elemen Scrumban tersebut dapat dilihat pada Tabel 2.4 (Bhavsar et al., 2020).

Tabel 2.4 Elemen Dasar Penyusun Scrumban

Framework	Elemen	Deskripsi
Sprint	Ritual Sprint	<i>Sprint planning, daily stand-up, dan retrospective</i>
	Sistem <i>pull</i> dan <i>push</i>	Prioritisasi pekerjaan untuk dilakukan
Kanban	Batasan WIP (<i>work in progress</i>)	Pembatasan pengambilan pekerjaan yang sedang berjalan
	CICD	Pengembangan yang dilakukan secara berkelanjutan, tidak menunggu satu siklus

Sumber: Bhavsar et al. (2020), telah diolah kembali

Beberapa dari ritual Sprint yang dilakukan adalah *sprint planning* untuk mempersiapkan rencana pekerjaan selama satu minggu ke depan, *daily stand-up* untuk menyampaikan kemajuan dalam pekerjaan, dan *retrospective* untuk kilas balik terhadap pekerjaan yang

telah dilakukan. Kemudian sistem *pull* dan *push* yaitu melakukan prioritisasi pekerjaan atau *task*. Dalam proses pengembangan, dibatasi pekerjaan yang dilakukan misalnya tiga buah *task* dalam seminggu. Apabila sebuah *task* sudah selesai dikerjakan, maka dapat langsung mengambil *task* baru selama tidak melebihi batasan WIP.

2.3 Arsitektur Sistem dan Data

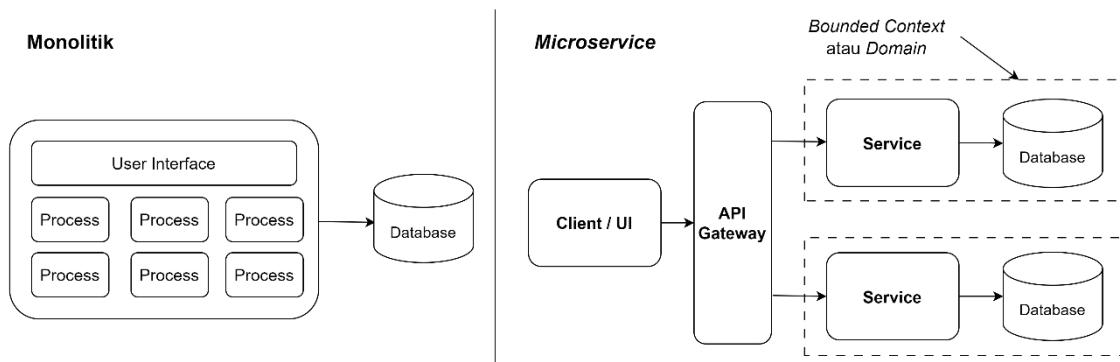
Aplikasi web pada umumnya dapat berjalan dengan bantuan teknologi pada sisi *server (backend)* dan *client (frontend)*. Implementasi aplikasi Mahoni pada penelitian ini hanya berfokus di sisi *server* dan bertujuan untuk membangun arsitektur yang sesuai dengan kebutuhan. Selain itu, dibutuhkan juga arsitektur data yang menunjang aplikasi Mahoni. Terdapat beberapa contoh dan istilah dalam arsitektur *backend* sistem dan data yaitu *microservice*, *event-driven*, dan *big data* yang dijelaskan pada beberapa subbab berikut.

2.3.1 Microservice

Desain arsitektur sebuah aplikasi tradisional selalu mengadopsi arsitektur monolitik di mana seluruh modul dan fitur berada pada suatu tempat yang sama. Arsitektur ini memiliki beberapa keunggulan yang di antaranya adalah kemudahan dalam pengembangan dan kesederhanaan karena komponen yang sedikit. Namun seiring berkembangnya waktu, sebuah aplikasi menjadi semakin besar dan kompleks yang menyebabkan munculnya beberapa masalah karena sebuah eror pada fitur tertentu dapat membuat seluruh aplikasi mati dan tidak bisa digunakan. Arsitektur monolitik juga membuat proses pengembangan aplikasi lebih sulit karena sebuah modul atau fitur sangat bergantung pada fitur lain sehingga proses *scaling* dan *deployment* tidak dapat dilakukan secara independen (Richards, 2015). Melihat dari kesulitan tersebut, muncul kebutuhan atas arsitektur yang tidak saling bergantung satu sama lain atau disebut dengan *decoupled*.

Microservice hadir sebagai salah satu solusi dalam merancang arsitektur yang *decoupled*. Arsitektur *microservice* adalah aplikasi yang dapat dilakukan *deployment*, *scale*, dan dites secara independen (Thones, 2015). Arsitektur ini dibangun atas dasar prinsip *service-oriented architecture* (SOA), di mana setiap servis bersifat *single-responsibility* yang berarti suatu servis hanya peduli pada fakta atau data yang berkaitan dengan domainnya sendiri. Konsep bahwa setiap servis mengatur data pada domainnya sendiri dikenal juga dengan istilah *bounded context* dan pertama kali muncul pada buku *Domain-Driven*

Design yang ditulis oleh Addison-Wesley pada tahun 2003. Sebuah domain adalah kumpulan komponen baik *database* atau komponen lain yang terenkapsulasi menjadi satu unit yang independen. Enkapsulasi ini memastikan bahwa hanya domain yang memiliki data saja yang dapat melakukan perubahan pada data tersebut. Gambar 2.4 (Richards, M., 2015) menjelaskan perbedaan topologi antara arsitektur monolitik dengan *microservice*.



Gambar 2.4 Perbedaan Topologi Arsitektur Monolitik dan *Microservice*

Sumber: *Software Architecture Patterns Understanding Common Architectural Styles and When to Use Them* (2015), telah diolah kembali

Pembagian menjadi beberapa domain yang independen dapat mempermudah proses pemeliharaan ketika aplikasi yang dibangun semakin kompleks dan meningkatkan *fault-tolerance* atau toleransi kegagalan sistem secara keseluruhan karena kegagalan pada suatu servis tidak mengganggu servis lain (Laigner et al., 2020). Namun demikian, arsitektur ini juga memiliki tantangan tersendiri seperti alur komunikasi yang menjadi lebih kompleks dan kesulitan dalam merancang desain *database*. Hal lainnya adalah jika diperlukan akses data antar domain, maka domain tersebut harus meminta dari domain terkait atau menggunakan cara lain seperti *trigger*, *materialized view*, dan lain-lain tanpa secara langsung mengakses *database*. Oleh karena itu, dibutuhkan metode atau arsitektur komplemen untuk menunjang komunikasi antar servis.

2.3.2 *Event-Driven*

Menurut Richards pada bukunya yang berjudul *Software Architecture Pattern*, arsitektur *event-driven* adalah arsitektur yang terdiri dari komponen-komponen yang bersifat ketidakhubungan tinggi (*highly decoupled*), bertujuan tunggal (*single-purpose*), dan secara asinkron menerima dan memproses *event*. Arsitektur ini terdiri dari beberapa

komponen utama yaitu: *event*, *event channel*, *subscriber*, dan *publisher*. Sebuah *event* adalah fakta yang menandakan telah terjadi sesuatu pada sistem (Stopford, 2018). Sebuah *event* dapat digunakan untuk memicu adanya aksi dari sebuah servis dan dapat juga digunakan sebagai metode perpindahan data antar servis (Umer et al., 2018). *Publisher* bertugas untuk membuat dan mengirim *event* ke sebuah *event channel*, kemudian *subscriber* pada *event channel* tersebut akan menerima seluruh *event* yang dikirim. *Message broker* bertugas untuk mengatur seluruh *event* yang masuk pada *event channel* dan mengirimkannya ke seluruh *subscriber* terkait.

Arsitektur ini dapat dibilang *loosely-coupled* karena setiap *publisher* yang mengirim *event* tidak tahu dan peduli siapa saja *subscriber*-nya (Umer et al., 2018). Di sisi lain, setiap *subscriber* pada *event channel* tertentu tidak tahu siapa yang membuat *event* tersebut. Arsitektur ini juga tidak sepenuhnya terpisah dengan arsitektur *microservice*. Sebuah servis dapat menjadi *publisher* dan *subscriber*. Arsitektur seperti ini disebut dengan *event-driven microservice* dan bersifat *eventually consistent* (Richardson, 2017). *Eventually consistent* berarti data di tempat yang berbeda pada akhirnya akan menjadi konsisten atau konvergen pada nilai yang sama, tetapi hal ini tidak menjamin kapan data tersebut akan konvergen (Kleppman, M, 2017).

2.3.2.1 Penelitian Terkait

Terdapat beberapa penelitian terkait yang menggunakan *event-driven* sebagai arsitektur utama sistem mereka. Penelitian pada tahun 2010 oleh Filippioni et al. menghasilkan sebuah sistem kota cerdas untuk *monitoring* area publik dengan sensor. Arsitektur *event-driven* dipilih karena dapat menghasilkan arsitektur yang fleksibel terhadap penambahan data sensor yang beragam dan komponen baru pada sistem.

Penelitian lain yang dilakukan oleh Winberg et al. pada 2021 mengembangkan suatu sistem untuk aplikasi kualitas udara secara *real-time* menggunakan arsitektur *event-driven* dengan Kafka sebagai *message broker*-nya. Bentuk arsitektur *event-driven* yang dipilih adalah *command-query-responsiblity segregation* (CQRS). Bentuk ini memisahkan proses mutasi data dengan proses pengambilan data pada servis yang berbeda.

Selain itu, terdapat pula penelitian oleh Laigner et al. pada 2020 yang melakukan perpindahan sistem padat data dari arsitektur monolitik ke arsitektur *event-driven*. Sistem yang dihasilkan terdiri dari beberapa *microservice* yang terhubung dengan Kafka sebagai *message broker*. Arsitektur *event-driven* yang dibangun memungkinkan sistem tersebut untuk bereaksi secara *real-time*, hal ini berguna untuk *monitoring* dan *alerts*. Kesimpulan yang diperoleh adalah pemisahan *microservice* dapat mempermudah proses pemeliharaan sistem dan adanya isolasi terhadap eror.

2.3.3 Big Data

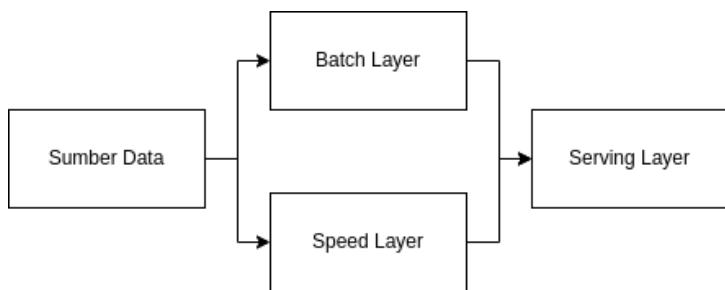
Penerapan kota cerdas dapat dibantu dengan pengolahan *big data*. *Big data* memegang peranan penting dalam penerapan kota cerdas untuk mengintegrasikan dan mengolah data-data yang bersumber dari IoT dan aktivitas masyarakat (Hashem et al., 2016). Secara umum terdapat dua kebutuhan saat membuat sebuah sistem *big data* yaitu, memproses data secara *real-time* dan melakukan analisis data dengan cepat (Feick et al., 2018). Kedua kebutuhan ini sejalan dengan penerapan kota cerdas yang mengandalkan kecepatan data dalam pengolahannya.

Big data merupakan kumpulan data yang memiliki 3 dimensi utama yaitu *Volume*, *Velocity*, dan *Variety* (Balusamy et al., 2021). *Volume* memiliki pengertian bahwa data yang diolah berukuran sangat besar dan berkembang sangat pesat setiap saat. *Velocity* memiliki pengertian bahwa data diproduksi dengan cepat serta membutuhkan pemrosesan serta analisis yang cepat. *Variety* memiliki arti bahwa format data yang tersedia memiliki banyak jenisnya yaitu *structured*, *semi-structured*, dan *unstructured*.

Dari ketiga dimensi utama tersebut, dimensi *velocity* merupakan dimensi yang paling sulit diatasi karena sebuah sistem yang mengolah *big data* harus memproses data secara *real-time* (Feick et al., 2018). Selain itu, *volume* data dari IoT dan aktivitas masyarakat juga meningkat akibat pertumbuhan masyarakat dan kebutuhan kota. Dengan demikian, diperlukan arsitektur *big data* yang terdiri dari beberapa *layer* pengolahan data untuk mengatasi permasalahan tersebut. *Layers* ini memiliki fungsi tersendiri dalam pemrosesan data sehingga dapat disimpan dan ditampilkan sebagai sebuah data analisis. Terdapat dua jenis arsitektur *big data* yang memiliki *layer*-nya masing-masing yaitu arsitektur Lambda dan Kappa.

2.3.3.1 Arsitektur Lambda

Arsitektur Lambda yang diperkenalkan oleh Nathan Mraz memiliki tiga *layer* penyusun yaitu *batch layer*, *streaming layer*, dan *serving layer* seperti pada Gambar 2.5 (Sanla et al., 2019). Pemrosesan data dilakukan pada dua *layer* pertama yaitu *batch layer* dan *speed layer*. *Speed layer* memproses data secara langsung sehingga data yang datang langsung diproses. Pada *layer* ini, data yang diproses dikawatirkan tidak lengkap dan tidak terproses dengan baik. Oleh karena itu terdapat *layer* untuk melengkapi data yaitu *batch layer*. *Batch layer* mengumpulkan data terlebih dahulu dalam bentuk *batch*, melakukan pemrosesan data, kemudian hasilnya digabungkan dengan data dari *speed layer*. Setelah itu, data-data tersebut disimpan untuk diolah kembali di *serving layer*.



Gambar 2.5 Arsitektur Lambda

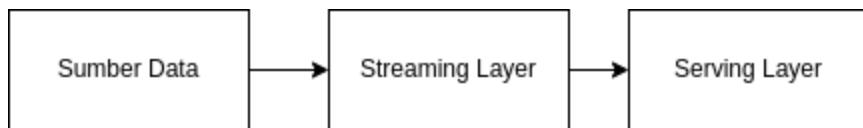
Sumber: Sanla et al. (2019), telah diolah kembali

Arsitektur ini memiliki kelebihan yaitu jaminan akurasi karena menggunakan dua *layer* dalam pemrosesan data yaitu *batch layer* dan *speed layer*. Dengan akurasi yang tinggi maka arsitektur ini memenuhi *fault tolerance* dengan adanya *batch layer* (Kalipe et al., 2019). Namun karena dalam pengumpulan data membutuhkan dua layer yang berbeda, arsitektur ini membutuhkan pemeliharaan atau *maintenance* untuk sinkronisasi data yang lebih rumit. Maka dari itu, untuk mengatasi masalah tersebut dikembangkan arsitektur lainnya yaitu arsitektur Kappa.

2.3.3.2 Arsitektur Kappa

Arsitektur Kappa memiliki dua *layer* yaitu *streaming layer* dan *serving layer* seperti pada Gambar 2.6 (Sanla et al., 2019). Arsitektur yang dikembangkan oleh Jay Kreps ini tidak memiliki *batch layer*. *Batch layer* tidak digunakan karena untuk mengatasi kekurangan

dari arsitektur Lambda yaitu *maintenance* dan sinkronisasi yang sulit akibat dua *layer* yang berbeda (Kalipe et al., 2019). Dengan memiliki satu *layer* saat melakukan pemrosesan data, arsitektur ini menjadi lebih sederhana dibandingkan arsitektur Lambda.



Gambar 2.6 Arsitektur Kappa

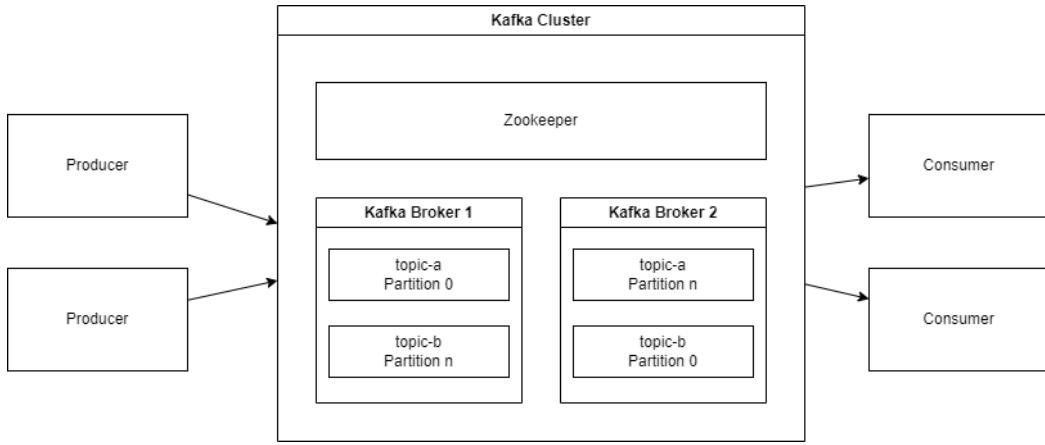
Sumber: Sanla et al. (2019), telah diolah kembali

Streaming layer berfungsi untuk melakukan pemrosesan data dan *serving layer* berfungsi untuk menyimpan dan melakukan *query* data untuk diolah kembali. Dengan hanya memiliki satu *layer* saja untuk pemrosesan data, harga untuk membangun arsitektur ini menjadi lebih rendah. Hal ini menguntungkan untuk membangun sebuah sistem kompleks yang membutuhkan fleksibilitas seperti sistem kota cerdas.

2.4 Apache Kafka

Dalam arsitektur *event-driven*, dibutuhkan *message broker* yang mengatur lalu lintas *event* baik kepada *publisher* maupun *subscriber*. Salah satu *message broker* yang populer digunakan adalah Apache Kafka. Apache Kafka adalah sebuah *message broker* yang dibuat oleh LinkedIn pada 2010 sebagai infrastruktur analitik dan solusi pemindahan data yang banyak dari *producer (publisher)* data yang berbeda ke *consumer (subscriber)* data yang juga berbeda (Magnoni, 2015).

Terdapat beberapa pilihan *message broker* lain yang dapat digunakan selain Apache Kafka. Perbedaan Kafka dengan *message broker* lain adalah bersifat terdistribusi dan berbasis *log*. *Message broker* berbasis *log* memiliki fokus kepada pencatatan dan penyimpanan *log*, yaitu segala sesuatu yang terjadi, untuk nantinya dapat dilihat kembali baik untuk keperluan *logging* ataupun analisis. Di samping itu, terdapat *message broker* berbasis *queue*, yang mana berfokus kepada komunikasi pesan dalam bentuk *queue* atau antrean. Penjelasan mengenai pemilihan *framework* dalam penelitian ini dijelaskan pada bab 5.



Gambar 2.7 Arsitektur Kafka *Cluster*

Sumber: Kafka: *The Definitive Guide* (2017), telah diolah kembali

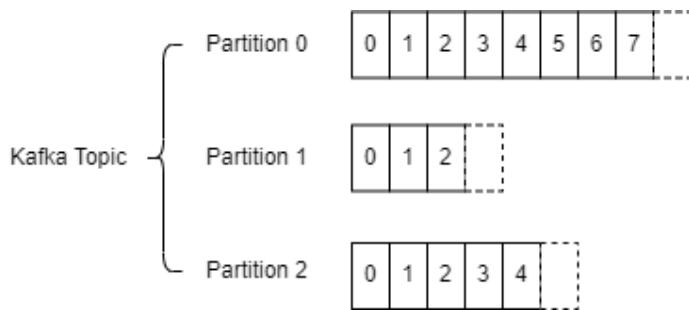
Gambar 2.7 (Narkhede et al., 2017) memperlihatkan arsitektur sebuah Kafka *cluster* yang terdiri dari dua *broker* Kafka dan sebuah Zookeeper. Zookeeper berfungsi sebagai *node* yang melakukan *leader election* dan *service discovery* antar *broker* Kafka. Zookeeper juga menyimpan data tentang *topic* apa saja yang dibuat pada suatu *cluster*. Kumpulan dari beberapa *broker* Kafka dan Zookeeper (versi terbaru Kafka tidak membutuhkan ini) disebut sebagai *cluster*. Jumlah *broker* yang banyak membuat Kafka memiliki performa yang tinggi karena memastikan sifat *fault-tolerance* dan memanfaatkan *horizontal scaling* yaitu meningkatkan skala dengan menaikkan jumlah *broker*.

2.4.1 Kafka Topic dan Partisi

Topic dalam Kafka merepresentasikan sebuah data *stream*. Setiap *topic* harus memiliki nama yang unik di dalam suatu *cluster* Kafka. Sebuah Kafka *producer* akan mengirim *event* ke *topic* dan Kafka *consumer* dapat memilih untuk *subscribe* ke *topic* tertentu. *Topic* di dalam Kafka dapat diatur nilai replikasinya. Setiap replikasi *topic* akan disimpan pada *broker* yang berbeda untuk mendukung sifat *fault-tolerance* Kafka, yaitu kemungkinan untuk mendeteksi kesalahan dan pulih dari kesalahan tersebut.

Setiap *topic* dibagi menjadi beberapa partisi atau *partition*. *Event* yang dimasukkan ke dalam partisi bersifat *immutable* atau tidak dapat diubah dan memiliki nilai *offset* yang menunjukkan posisi *event* pada partisi tersebut. Nilai *offset* ini digunakan untuk mengingat *event* terakhir yang telah diproses oleh masing-masing Kafka *consumer*. Kafka

memastikan bahwa setiap *event* pada sebuah partisi pasti terurut. Hal ini sangat berguna dalam skalabilitas sistem karena dapat dibuat beberapa *consumer* yang memproses secara paralel terhadap masing-masing partisi untuk mengatasi jumlah *event* yang banyak dari sensor-sensor yang ada di sistem kota cerdas. Gambar 2.8 (Narkhede et al., 2017) memberikan ilustrasi *topic* yang memiliki beberapa partisi.



Gambar 2.8 Representasi *Log* pada Sebuah Kafka *Topic* dengan Beberapa Partisi

Sumber: *Kafka: The Definitive Guide* (2017), telah diolah kembali

Setiap *topic* juga dapat diatur *retention type* yang menentukan bagaimana Kafka mempertahankan *event* di dalam *log*. Terdapat dua opsi yang dapat dipilih, yaitu *time-based* dan *key-based*. Jika sebuah *topic* diatur untuk memiliki *time-based retention type*, maka setiap *event* dalam *topic* tersebut disimpan untuk durasi waktu tertentu kemudian dihapus. Tipe ini berguna untuk menyimpan data historis (*Streams Concepts*, n.d.).

The diagram shows two tables representing a Kafka topic's log. The left table has four rows (Offset 0-3) and columns for Offset, Key, and Value. The right table has two rows (Offset 2-3) and columns for Offset, Key, and Value. Arrows point from the third row of the left table to the second row of the right table, indicating that the entry at offset 3 was retained while the entry at offset 2 was deleted.

Offset	Key	Value
0	K1	V1
1	K2	V2
2	K1	V3
3	K2	V4

Offset	Key	Value
2	K1	V3
3	K2	V4

Gambar 2.9 Key-Based Retention Type Topic

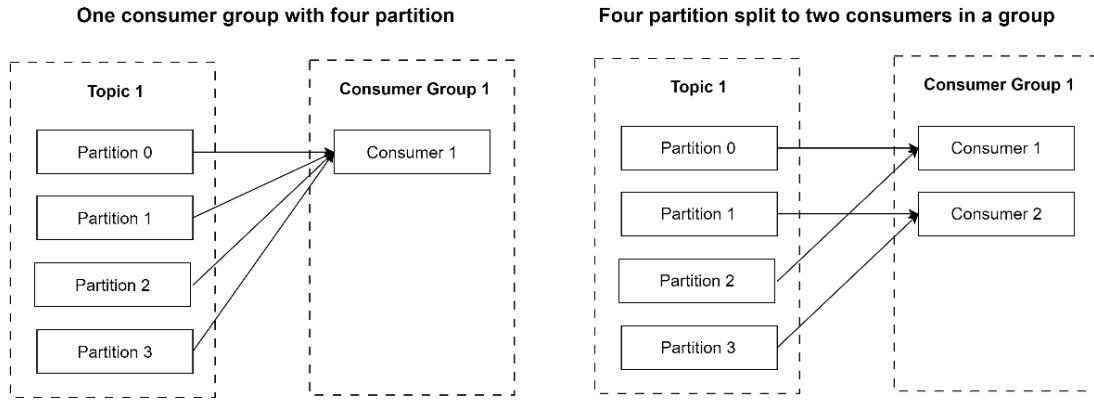
Sumber: Apache Kafka documentation. (n.d.). Retrieved May 28, 2023, telah diolah kembali

Berbeda dengan *time-based, key-based retention type* akan mengganti *event* yang memiliki *key* yang sama. Hal ini memastikan bahwa hanya nilai terakhir saja yang disimpan di dalam Kafka. Gambar 2.9 (*Apache Kafka Documentation*, n.d.) memberikan ilustrasi *compaction* pada *key-based retention type topic*. *Topic* yang menggunakan *retention type* ini sering juga disebut sebagai *compacted topic*. Tipe ini dapat menghemat penyimpanan karena jumlah *event* hanya terbatas pada banyaknya jumlah *key* dan digunakan untuk proses *Change Data Capture*. Jaminan yang disediakan oleh *key-based retention* adalah (1) semua *consumer* yang sudah mendengarkan *offset* terakhir akan mendapatkan seluruh *event* yang dikirim; (2) semua *consumer* yang sudah mendengarkan *offset* terakhir akan mendapatkan seluruh *event* yang dikirim; (3) urutan *event* akan dipertahankan; (4) nilai *offset* sebuah *event* tidak akan berubah; (5) setiap *consumer* baru yang mendengarkan *topic* akan mendapatkan nilai terakhir untuk setiap *key* (*Apache Kafka Documentation*, n.d.).

2.4.2 Kafka Consumer dan Kafka Producer

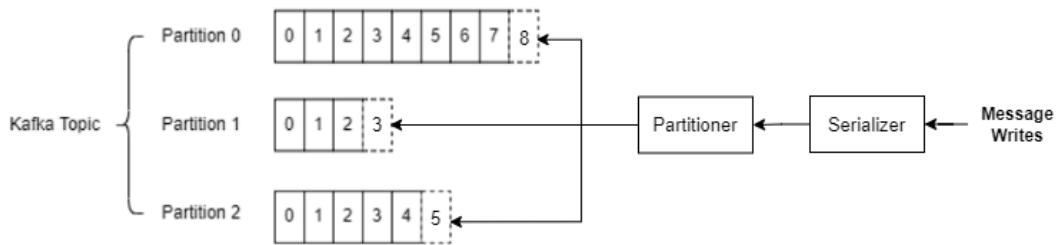
Terdapat dua jenis *client (event processor)* di Kafka yaitu *producer* dan *consumer*. Pada dasarnya sebuah *event* dikirim oleh *producer* dan diterima oleh *consumer* di sebuah *topic* tertentu. Namun, terdapat beberapa perbedaan perilaku jika dibandingkan dengan *publisher* dan *subscriber* biasa yang menyebabkan Kafka dapat memiliki sifat *high availability* dan *high scalability*. Setiap *consumer* merupakan bagian dari sebuah *consumer group*. Ketika terdapat beberapa *consumer* yang melakukan *subscribe* ke suatu *topic* dan merupakan bagian dari *consumer group* yang sama, maka masing-masing *consumer* hanya akan menerima data dari partisi yang berbeda seperti yang diilustrasikan pada Gambar 2.10 (Narkhede et al., 2017). Perilaku ini memungkinkan Kafka untuk dapat memiliki sifat *high scalability* dan performa yang tinggi dengan membagi beban ke beberapa *consumer* yang berbeda secara paralel.

Producer pada Kafka menulis *event* ke sebuah partisi di *topic*. Sebelum dikirimkan ke *topic*, *event* tersebut akan diserialisasi terlebih dahulu. Selanjutnya dilakukan penentuan partisi secara otomatis menggunakan algoritma *round-robin* atau secara manual dengan menentukan partisi saat melakukan pengiriman *event*. Alur pengiriman *event* dapat dilihat pada Gambar 2.11 (Narkhede et al., 2017).



Gambar 2.10 Pembagian Beban antar *Consumer Group* yang *Subscribe* ke Sebuah *Topic*

Sumber: Kafka: *The Definitive Guide* (2017), telah diolah kembali



Gambar 2.11 Alur Pengiriman *Event* oleh Kafka *Producer*

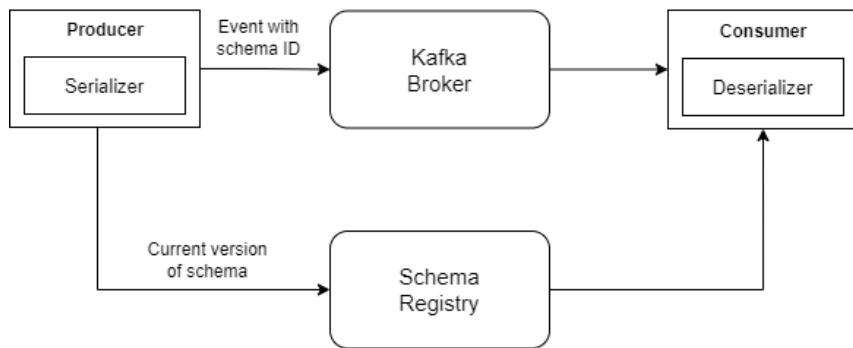
Sumber: Kafka: *The Definitive Guide* (2017), telah diolah kembali

Perlu diingat bahwa Kafka hanya menjamin urutan untuk setiap partisi. Oleh karena itu, penentuan partisi ini penting karena akan menentukan apakah *event* yang dikirim dapat dikonsumsi secara berurutan atau tidak pada *consumer* yang berbeda.

2.4.3 Serialisasi dan Kafka *Schema Registry*

Setiap data yang dikirim ke Kafka harus diserialisasi terlebih dahulu. Proses serialisasi dilakukan menggunakan *serializer* yang disediakan oleh Kafka atau *serializer* lain seperti Avro, Thrift, dan Protobuf. Untuk *event* yang kompleks disarankan untuk membuat sebuah *schema* agar *event* dapat dikirim dan dikonsumsi secara konsisten. Setiap *schema* harus disimpan pada sebuah *schema registry* yang dijalankan terpisah dari *broker* Kafka. *Schema registry* sendiri bukan merupakan bagian dari Apache Kafka, tetapi terdapat beberapa implementasi *open source* seperti yang ditawarkan oleh Confluent Schema

Registry. Gambar 2.12 (Narkhede et al., 2017) menjelaskan alur serialisasi *event* pada Kafka.



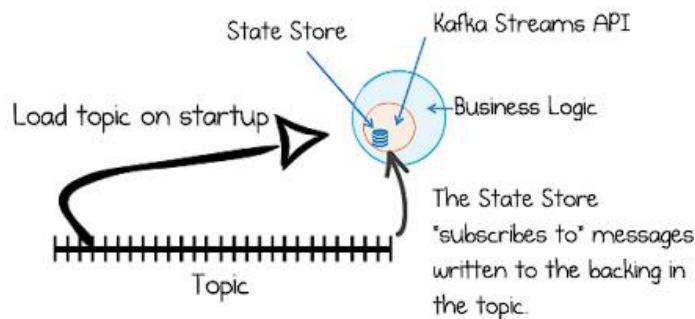
Gambar 2.12 Alur Serialisasi Menggunakan *Schema Registry*

Sumber: *Kafka: The Definitive Guide* (2017), telah diolah kembali

Schema registry dibuat terpisah dari Kafka untuk memastikan bahwa performa Kafka tidak menurun dan tetap memiliki *throughput* yang tinggi. Sebelum Kafka *producer* mengirim *event* ke *topic* di Kafka, *event* tersebut akan divalidasi terlebih dahulu oleh *schema registry*. Bila bentuk *event* sesuai, baru akan dikirim ke Kafka. Sebaliknya, Kafka *consumer* akan meminta detail *schema* dari *schema registry* untuk setiap *topic* yang dikonsumsi. *Schema registry* juga bersifat terdistribusi dan dapat dibuat lebih dari satu *node* dalam sebuah *cluster*.

2.4.4 KTable

KTable merupakan abstraksi tingkat tinggi dari sebuah *compacted topic* (*key-based retention type*) di Kafka. Abstraksi tingkat tinggi ini secara otomatis akan melakukan *subscribe* ke sebuah *topic*, membaca seluruh *event* dalam *topic* tersebut, dan menyimpan hasilnya ke sebuah penyimpanan atau *state store* yang disimpan secara lokal oleh Kafka Streams API. KTable memungkinkan akses terhadap *compacted topic* layaknya sebuah *key-value database*. Gambar 2.13 (Stopford, B., 2018) menggambarkan bagaimana *state store* di Kafka dapat diperoleh dari *event* dalam *topic*.



Gambar 2.13 State Store di Kafka Streams API Menggunakan KTable

Sumber: *Designing Event-Driven Systems Concepts and Patterns for Streaming Services with Apache Kafka* (2018)

State store pada KafkaStreams bersifat *fault tolerance*. Bila terjadi sesuatu yang menyebabkan *data corruption*, KafkaStreams dapat membuat ulang isi dari *state store* dengan melihat kembali isi dari *compacted topic*.

2.5 Stream Processing

Arsitektur *big data* yang diterapkan oleh kota cerdas mendapatkan data dari berbagai sumber, salah satunya dari IoT. Data yang bersumber dari IoT merupakan data yang tidak terbatas (*unbounded*) atau bersifat *stream* karena mempunyai sifat *real-time*. Untuk mengolah data ini diperlukan sebuah *framework stream processing* yang dapat mengolah data *stream*. *Framework stream processing* memiliki beberapa konsep yang harus diperhatikan yaitu waktu pengolahan dan penggabungan data.

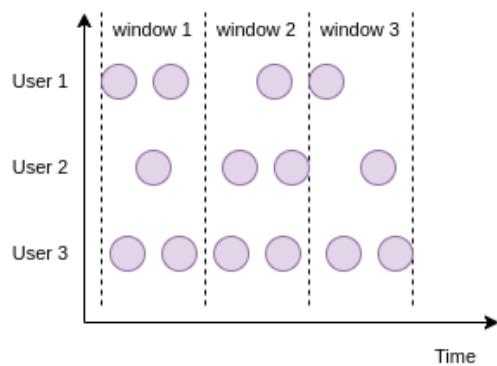
2.5.1 Windowing

Data *stream* bergantung pada penggunaan waktu yang dipakai. Terdapat dua jenis penggunaan waktu yaitu *event time* dan *processing time* (Kleppmann, M., 2018). Penggunaan waktu berdasarkan *event time* menggunakan *timestamp* saat data tersebut terjadi, sedangkan *processing time* menggunakan *timestamp* saat data tersebut diproses.

Penggunaan waktu dengan *event time* harus berdasarkan satuan waktu yang sama. Menggunakan satuan waktu yang berbeda bisa menyebabkan arti yang berbeda saat menggunakannya. Maka dari itu terdapat beberapa cara untuk mengatasi hal tersebut yaitu, waktu saat *event* terjadi berdasarkan waktu *device*, waktu saat mengirimkan *event*

berdasarkan waktu *server*, dan waktu saat menerima *event* berdasarkan waktu *server*. Perbedaan penggunaan waktu mempengaruhi proses *windowing* dalam memproses data *stream*.

Windowing merupakan cara untuk menangkap data berdasarkan penggunaan waktu dalam rentang tertentu. Data *stream* yang tidak terbatas akan dipecah-pecah menjadi beberapa *batch* berdasarkan waktu yang menjadikan data mempunyai batas akhir. Secara umum terdapat beberapa macam *windowing* yaitu *tumbling window*, *sliding window*, dan *session window* (Windows, n.d.).

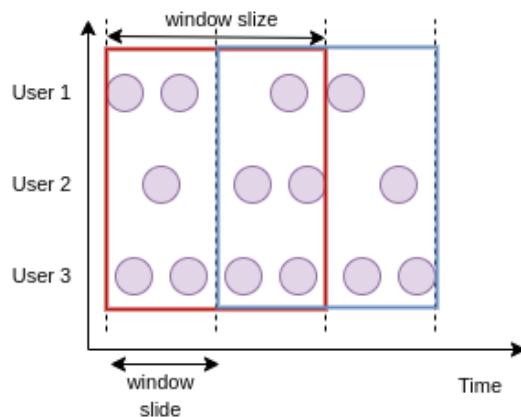


Gambar 2.14 Tumbling Window

Sumber: Windows (n.d.), telah diolah kembali

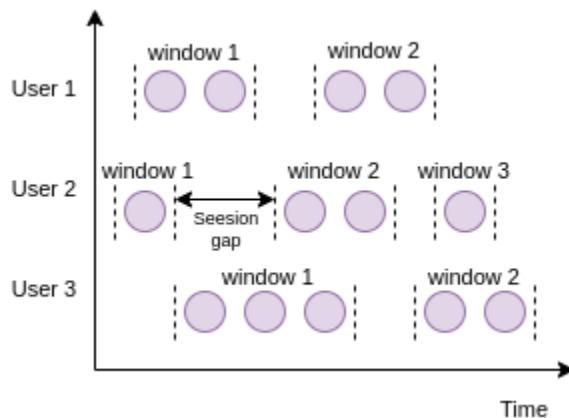
Tumbling window yang dapat dilihat pada Gambar 2.14 (Windows, n.d.) merupakan *windowing* yang menggunakan rentang waktu yang tetap dan tidak memperbolehkan adanya *overlapping* dalam mengambil data. Waktu yang diatur dibulatkan ke waktu yang terdekat sehingga tidak ada *overlapping*. *Windowing* ini biasanya digunakan untuk mendapatkan data dalam rentang waktu yang tetap namun tidak mementingkan detail yang terjadi pada rentang waktu yang lebih kecil.

Sliding window memiliki rentang waktu yang tetap sama seperti *tumbling window*. Seperti yang dapat dilihat pada Gambar 2.15 (Windows, n.d.), *sliding window* memperbolehkan adanya *overlapping* yang menjadi *window size*. Namun *sliding window* memiliki kontrol untuk secara teratur mengatur rentang waktu *window size* yang lebih kecil dari ukuran *window* utama. Hal ini digunakan untuk mengetahui detail data yang ditangkap dalam rentang waktu *sliding* tersebut selama waktu ukuran *window* utama.



Gambar 2.15 Sliding Window

Sumber: Windows (n.d.), telah diolah kembali



Gambar 2.16 Session Window

Sumber: Windows (n.d.), telah diolah kembali

Berbeda dengan kedua *windowing* sebelumnya, *session window* merupakan *windowing* yang tidak terikat oleh waktu yang tetap. Gambar 2.16 (Windows, n.d.) menunjukkan bahwa *windowing* ini terikat pada sebuah waktu *session* ketika sebuah *event* terjadi. Sebuah *event* memiliki *session gap*-nya tersendiri, sehingga tidak ada pengaturan waktu yang ditentukan untuk seluruh *event*. Teknik ini cocok untuk menghitung *event* yang terjadi ketika data memiliki rentang waktu tertentu saat aktif dan tidak aktif.

Kemampuan *windowing* yang dimiliki oleh *framework stream processing* memudahkan penggunanya untuk memproses data dengan penggunaan tertentu. Sering kali *data stream*

diproses dalam rentang waktu tertentu untuk dilakukan perhitungan seperti rata-rata, nilai maksimum, dan nilai minimum serta pemrosesan data lainnya dalam rentang waktu tertentu. Perhitungan atau pemrosesan data tersebut membutuhkan komputasi yang lebih besar dan rumit jika *framework stream processing* tidak memiliki mekanisme *windowing*.

2.5.2 Stream Join

Sumber data *stream* yang diproses di dalam sebuah sistem dapat berasal lebih dari satu. Kumpulan data dari berbagai sumber data ini dapat digabungkan untuk saling melengkapi dengan menggunakan metode *join*. Metode *join* dalam *stream processing* dibagi berdasarkan sumber data yang digunakan. Setiap metode memiliki cara yang berbeda-beda dalam melakukan prosesnya masing-masing. Metode tersebut di antaranya *Stream-Stream join*, *Stream-Table join*, dan *Table-Table join* (Kleppmann, M., 2018).

Stream-Stream join dilakukan dengan menggabungkan beberapa data *stream* dari beberapa sumber yang berbeda. Data *stream* ini diproses dengan menggunakan sebuah *window* atau batas waktu tertentu berdasarkan parameter yang telah ditentukan. Karena proses *join* ini menggunakan *window*, maka *join* ini sering disebut sebagai *window join*. Proses *Stream-Stream join* efektif digunakan ketika sebuah *stream processing* menerima data dari beberapa IoT atau data lain yang sifatnya *stream* untuk digabungkan menjadi data yang lebih lengkap.

Berbeda dengan *Stream-Stream join*, metode *Stream-Table join* dilakukan dengan menggabungkan data *stream* dengan data *table*. Penggabungan ini dilakukan untuk melengkapi data *stream* dengan informasi tambahan dari data *table*. Data *stream* biasanya tidak memuat informasi dengan detail karena ukurannya yang kecil. Berbeda dengan data *stream*, data *table* memiliki data yang lebih lengkap karena tersimpan pada suatu *database* dengan ukuran tertentu. Proses melengkapi data *stream* dengan data *table* disebut sebagai proses *data enrichment*.

Terdapat dua cara untuk melakukan *Stream-Table join* yaitu dengan melakukan *query ID* pada *table database* utama ketika menerima data atau dengan menyalin data *table*. *Query* merupakan cara untuk mengambil data dari sebuah *database* berdasarkan parameter tertentu salah satunya dengan ID. ID merupakan nilai unik setiap data yang disimpan di dalam *table*. Dengan melakukan *query ID* pada *table* utama, proses komputasi menjadi

lebih besar sehingga dapat mengganggu kinerja *database*. Alternatif lainnya adalah dengan menyalin isi dari *table database* utama. Hal ini dapat meminimalkan gangguan pada *table* utama. Selain itu, dengan menerapkan hal ini sistem memiliki *backup* data apabila terjadi masalah. Salah satu cara menyalin isi *data table* utama adalah dengan melakukan *Change Data Capture* (CDC) yang dijelaskan pada subbab 2.7.

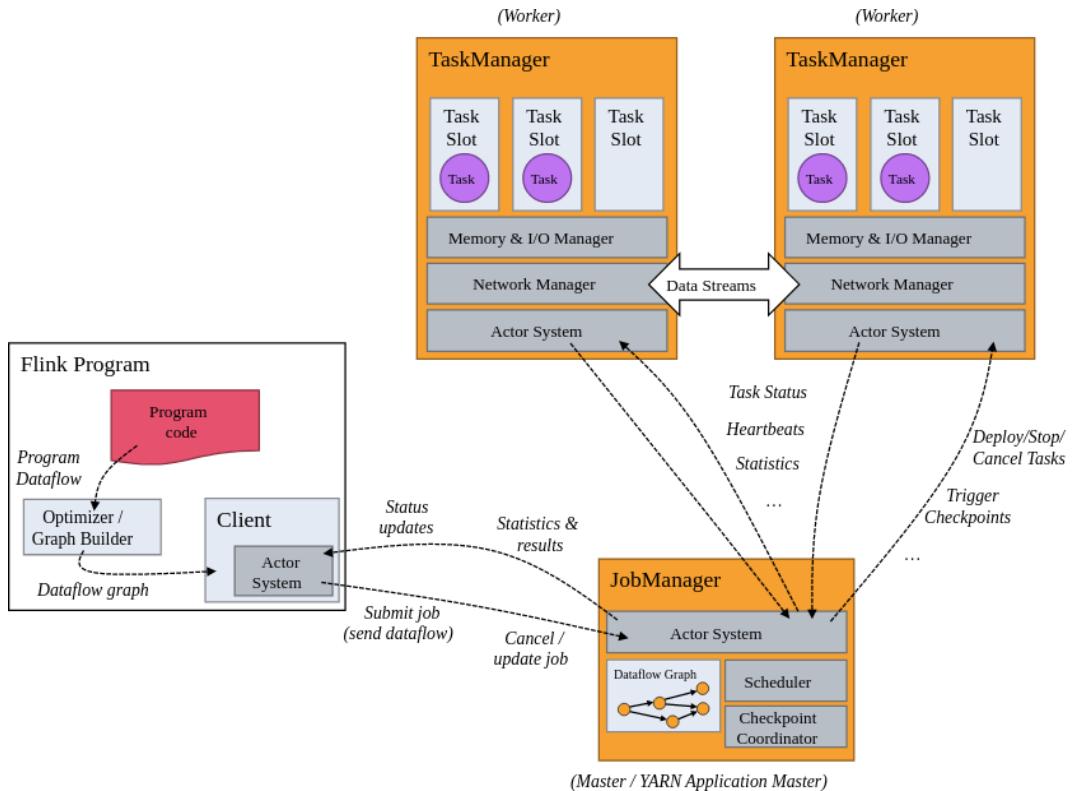
Pada dasarnya, *Stream-Table join* dan *Stream-Stream join* memiliki kemiripan, tetapi hal yang membedakannya adalah sumber data dan penggunaan *window* pada *Stream-Stream join*. *Stream-Stream join* membutuhkan batasan rentang waktu agar data dapat memiliki batasan data untuk melakukan *join* yang jelas akibat data yang bersifat tidak terbatas. Akan tetapi, *Stream-Table join* tidak membutuhkan hal tersebut karena *data enrichment* memiliki batasan yang jelas seperti data ID.

Selanjutnya terdapat *Table-Table join* yang digunakan dalam konteks sebagai *cache table* yang terhubung dengan data *stream*. *Cache* merupakan tempat penyimpanan data sementara yang digunakan atau diakses oleh pengguna sehingga tidak membutuhkan akses ke *database* secara langsung. *Table-Table join* ini berguna untuk melakukan penggabungan data yang tersimpan di dalam *cache* sehingga hasil *stream processing* dapat diakses dan digunakan oleh pengguna.

2.5.3 Apache Flink

Apache Flink merupakan salah satu *framework* yang digunakan untuk *stream processing*. Terdapat beberapa pilihan *framework* lain yang dapat digunakan selain Apache Flink yaitu Spark dan Kafka Stream. Penjelasan mengenai pemilihan *framework* yang digunakan dalam penelitian ini dijelaskan pada bab 6.

Untuk mendukung pemrosesan data *stream*, Flink memiliki arsitektur tersendiri yang terdiri dari komponen *master* berupa JobManager dan *worker* berupa TaskManager yang terlihat pada Gambar 2.17 (Flink architecture, n.d.). Komponen *master* merupakan komponen yang bertugas untuk mendelegasikan tugas kepada komponen *worker* yang berfungsi untuk mengeksekusi tugas tersebut. Tugas atau *job* yang dikerjakan oleh Flink ditulis dengan menggunakan bahasa pemrograman Java atau Scala dalam bentuk *file JAR*. *File* ini diterima oleh JobManager yang akan mengubahnya menjadi JobGraph. JobGraph merupakan hasil penguraian *job* yang digunakan untuk pemrosesan data.



Gambar 2.17 Arsitektur Flink

Sumber: Flink architecture (n.d.)

JobManager terdiri dari beberapa komponen yaitu ResourceManager, Dispatcher, dan JobMaster. JobMaster merupakan komponen yang bertugas untuk mengubah kode menjadi JobGraph. Setelah diubah menjadi JobGraph, JobMaster akan mengatur *job* tersebut untuk dikerjakan oleh TaskManager. JobMaster mengatur Task Slot yang tersedia di TaskManager sehingga *job* yang dikerjakan dapat teralokasi dengan baik (Flink architecture, n.d.). *Job* yang sudah dialokasikan ke TaskManager dapat dipecah menjadi beberapa bagian atau *task* yang dikerjakan oleh masing-masing Task Slot yang berbeda secara paralel. Hal ini akan meringankan kinerja TaskManager sehingga *job* dapat diproses dengan cepat.

Dengan komponen dan kemampuan Flink yang dapat mengatur pemrosesan data secara mandiri, pemrosesan data dapat menjadi lebih mudah. *Developer* hanya perlu mengatur berapa banyak komponen yang dibutuhkan seperti TaskManager dan jumlah Task Slot yang disediakan oleh masing-masing TaskManager. Selain itu, pembagian serta

pengerjaan *job* yang dilakukan secara paralel membuat Flink menjadi lebih ringan dan cocok untuk mengolah data *stream*.

2.6 Penyimpanan Data

Pada penelitian ini digunakan beberapa jenis penyimpanan data untuk mengakomodasi berbagai fungsi maupun kebutuhan. Fungsi yang digunakan untuk operasional servis dan pengolahan *big data* memiliki kebutuhan yang berbeda. Beberapa jenis *database* yang digunakan yaitu *relational database*, *non-relational database*, dan *data warehouse* yang dijelaskan pada subbab di bawah.

2.6.1 Relational Database

Relational database merupakan *database* yang digunakan untuk menyimpan data terstruktur sesuai dengan skema data yang disimpan. Jenis *database* ini lazim digunakan untuk menyimpan data pengguna pada servis sebuah sistem aplikasi karena memiliki struktur yang jelas. Struktur data yang jelas membuat *database* ini memiliki kemampuan untuk melakukan *query* yang rumit.

Relational database memiliki sifat ACID (*Atomicity*, *Consistency*, *Isolation*, dan *Durability*) (Kleppmann, M., 2018) yang memastikan bahwa kinerja *database* saat diakses berjalan dengan baik. *Database* yang diakses oleh berbagai pengguna dalam waktu dan rentang yang berbeda harus memiliki data yang konsisten dan sama (*Consistency* dan *Isolation*). Konsistensi data juga berlaku saat sistem tidak mengalami eror maupun saat terjadi eror. Saat data sukses untuk ditambahkan atau diubah, maka data tidak dapat dikembalikan lagi (*Durability*). Namun saat terjadi eror seperti *crash* pada sistem, semua permintaan perubahan data akan ditolak agar data tetap konsisten seperti saat sebelum terjadi eror (*Atomicity*).

Konsistensi yang harus dijaga pada *relational database*, membuat *database* ini tidak cocok digunakan untuk menyimpan data semi-terstruktur dan tidak terstruktur. Data jenis ini biasa digunakan untuk data *stream* yang mengandalkan kecepatan data. Oleh karena itu, terdapat jenis *database* lain yang digunakan pada penelitian ini untuk menyimpan data *stream* yaitu *non-relational database* atau NoSQL.

2.6.2 Non-Relational Database (NoSQL)

Non-relational database umumnya digunakan untuk menyimpan data yang tidak terstruktur dan diakses dengan cepat. Berbeda dengan *relational database* yang menjamin konsistensi data dengan ACID, *non-relational database* tidak menjamin konsistensi tersebut. Menurut Brewer, NoSQL memiliki sifat BASE (*Basically Available, Soft state, dan Eventually consistent*) yang merupakan lawan dari ACID (Lourenço, J. R. et al., 2015). NoSQL tidak menjamin konsistensi data setiap saat (*Soft state* dan *Eventually consistent*), namun NoSQL mudah untuk diakses (*Basically Available*) sehingga cocok digunakan oleh arsitektur *big data* yang membutuhkan *database* yang dapat diakses dengan cepat.

Dalam penelitian ini, digunakan dua *framework non-relational database* untuk menyimpan data secara *real-time* yaitu InfluxDB dan Cassandra. Setiap *framework* tersebut memiliki tujuan dan kemampuan yang berbeda dan dijelaskan pada subbab ini.

2.6.2.1 InfluxDB

InfluxDB merupakan *database* NoSQL dengan jenis *time series*. *Database* ini menyimpan data berdasarkan waktu atau *timestamp*. Waktu merupakan salah satu komponen dari data transaksional yaitu data yang mencatat aktivitas interaksi pengguna dengan aplikasi. Data seperti ini digunakan untuk membuat analisis data seperti visualisasi data yang dapat menggambarkan aktivitas pengguna.

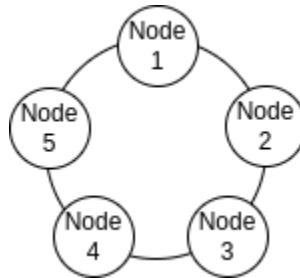
Semua data InfluxDB disimpan di dalam sebuah *bucket*. Di dalam *bucket* tersebut, data disimpan di dalam sebuah tabel yang memiliki beberapa kolom yaitu **_time**, **_measurement**, **_tag**, **_field**, dan **_value** (InfluxDB Data elements, n.d.). Kolom **_measurement** diisi dengan nama yang mendeskripsikan data secara umum atau dapat dianalogikan dengan nama tabel pada *relational database*. *Timestamp* data disimpan di dalam kolom **_time** dan data lainnya disimpan di dalam **_tag**, **_field**, dan **_value**. Data yang tidak unik dapat disimpan di dalam **_tag** sedangkan data yang unik tetapi bukan sebagai identifikasi pembeda data dapat disimpan di dalam **_field** dan **_value**.

Penyimpanan data yang sesuai fungsinya tersebut serta menyimpan berdasarkan waktu membuat data dapat diproses menjadi visualisasi data dalam bentuk grafik waktu. Tidak

hanya InfluxDB yang merupakan *time series database framework*, terdapat beberapa pilihan yang dapat digunakan yang dijelaskan pada bab 6.

2.6.2.2 Cassandra

Cassandra merupakan NoSQL *database* yang terdistribusi. Seperti pada Gambar 2.18 (Cassandra Basics, n.d.), Cassandra dapat terdiri dari beberapa *node*. *Node* merupakan sebuah mesin yang dalam konteks ini adalah *database*. Penyimpanan data yang terdistribusi memiliki pengertian bahwa *database* Cassandra dapat dikembangkan atau *scaling* dengan menambah *node* yang ada. *Nodes* Cassandra dapat saling berkomunikasi satu sama lain untuk menyimpan data. Kemampuan komunikasi antar *node* membuat Cassandra dapat mereplikasi data di semua *node*.



Gambar 2.18 Arsitektur Cassandra

Sumber: Cassandra Basics (n.d.), telah diolah kembali

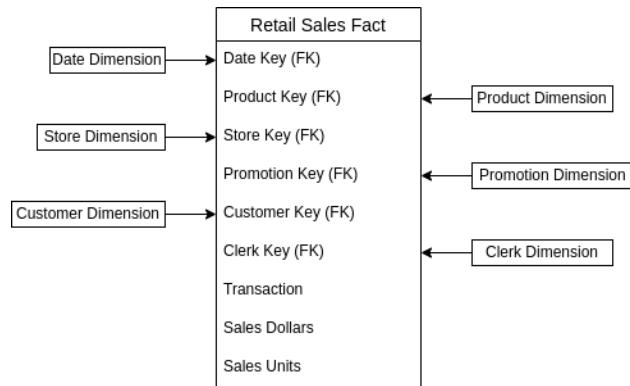
Kemampuan replikasi ini membuat Cassandra dapat selalu tersedia untuk diakses, sehingga mengurangi kemungkinan-kemungkinan eror yang dapat terjadi akibat *database* yang tidak dapat diakses. Selain itu, data yang disimpan juga dapat dipartisi ke beberapa *node* yang berbeda. Hal ini membuat data dapat terdistribusi dengan baik sehingga tidak memberatkan satu *node* saja.

2.6.3 Data Warehouse

Data warehouse merupakan *database* yang menyimpan data dalam jumlah yang besar. Pada penelitian ini *data warehouse* menyimpan semua data baik data transaksional maupun data non-transaksional. Seperti yang sudah dijelaskan sebelumnya, data transaksional merupakan data aktivitas pengguna yang lebih dinamis sedangkan data non-transaksional merupakan data yang menyimpan informasi yang lebih statis. *Data*

warehouse dapat juga berfungsi sebagai *single version of truth* (Sen, A. et al., 2012). Fungsi tersebut memiliki pengertian bahwa semua data pada sebuah sistem disimpan di tempat yang sama, sehingga baik itu data baru maupun data lama tersimpan di dalamnya.

Penyimpanan semua data di satu tempat yang sama membuat *data warehouse* dapat dijadikan sumber data untuk membuat sebuah laporan analitik. Pembuatan laporan analitik membutuhkan *database* yang mampu mengolah data dalam jumlah besar dan kompleks. *Database* yang digunakan tidak bisa menggunakan OLTP (*Online Transaction Processing*) *database* biasa, tetapi harus menggunakan *data warehouse* yang dapat menjalankan OLAP (*Online Analytic Processing*) (Kleppmann, M., 2018). OLTP *database* merupakan *database* yang digunakan untuk memproses data pengguna seperti *relational database*. *Database* ini memiliki kemampuan *query* yang lebih sederhana, menyimpan data terkini, dan memiliki ukuran data sampai *Terabyte*. Di sisi lain, *data warehouse* dapat menjalankan OLAP yang merupakan kemampuan untuk memproses data analitik yang menggunakan data historis, *query* yang kompleks, dan memiliki ukuran data sampai *Petabytes*. Oleh karena itu, proses analitik mengambil data dari *data warehouse* sebagai sumbernya sehingga proses pengambilan data tidak mengganggu kinerja OLTP *database*.



Gambar 2.19 Star Schema

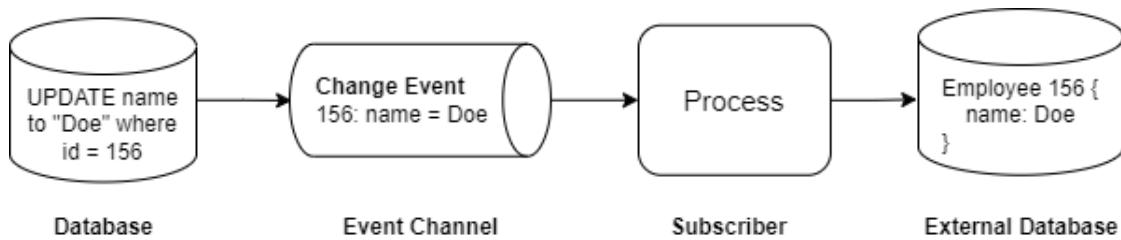
Sumber: Kimball, R., & Ross, M. (2013.), telah diolah kembali

Data warehouse pada umumnya menyimpan data dalam bentuk *dimensional modeling*. Terdapat dua jenis tabel pada *dimensional modeling* yaitu *fact table* dan *dimension table* (Kimball, R., & Ross, M., 2013). *Fact table* berisi data yang dihasilkan oleh transaksi-

transaksi yang ada pada sistem sedangkan *dimension table* berisi data yang menjelaskan komponen penyusun *fact table* lebih detail. Untuk menyimpan *dimensional modeling* ini, terdapat satu bentuk skema yang umum digunakan yaitu *star schema*. *Star schema* memiliki satu *fact table* di tengah dengan memiliki *dimension table* yang menyusun *fact table* tersebut seperti pada Gambar 2.19 (Kimball, R., & Ross, M., 2013).

2.7 Change Data Capture (CDC)

Fitur *real-time monitoring* pada aplikasi Mahoni memerlukan ketersediaan data secara cepat untuk proses *enrichment*. Kebutuhan ini membuat perlunya replikasi data transaksional aplikasi mahoni ke sebuah *database* baru yang dapat diakses lebih cepat. Replikasi data dari suatu sumber ke tempat lain dapat dilakukan dengan menggunakan sebuah *log* untuk mencatat seluruh perubahan pada data. Konsep ini merupakan ide dari *change data capture* (Kleppmann, 2016). Contoh alur CDC dapat dilihat pada Gambar 2.20 (*What is Change Data Capture? How Does It Work?*, n.d.).



Gambar 2.20 Change Data Capture

Sumber: *What is Change Data Capture? How Does It Work?* (n.d.), telah diolah kembali

Implementasi CDC pada aplikasi Mahoni memindahkan data dari *database PostgreSQL* menuju Cassandra dan *data warehouse*. Salah satu solusi implementasi CDC adalah dengan menggunakan bantuan alat lain seperti Debezium dan Kafka Connect. Kafka Connect merupakan *framework* untuk menghubungkan Kafka dengan sumber data lain dengan bantuan *source connector* dan *sink connector*. Sementara itu, Debezium adalah salah satu implementasi *source connector* yang dapat menghubungkan Kafka dengan *database* secara langsung (*Debezium Architecture*, n.d.). Terdapat beberapa *database* yang didukung oleh Debezium antara lain PostgreSQL, MySQL, MongoDB, Cassandra, dan masih banyak lagi. CDC pada *database PostgreSQL* memanfaatkan sebuah *log* yang dinamakan *write ahead log* (WAL) dan *logical decoding*. WAL pada PostgreSQL

mencatat seluruh perubahan data secara *low-level*, sedangkan *logical decoding* membaca WAL dan mengubahnya menjadi *events* yang dapat diolah. WAL inilah yang dibaca oleh *source connector* Debezium untuk menghasilkan *events* yang dikirimkan ke Kafka. Di sisi lain, terdapat juga implementasi Cassandra dan Google BigQuery *sink connector* yang akan membaca *event* yang ada di Kafka dan memindahkannya ke dalam *database* tersebut.

BAB 3

METODE DAN METODOLOGI UMUM

Bab ini menjelaskan metode-metode yang digunakan untuk menyelesaikan sub masalah yang diidentifikasi dalam penelitian ini sesuai dengan tiga pertanyaan penelitian yang sudah dijelaskan pada subbab 1.2. Selain itu, terdapat metodologi umum yang dapat mengevaluasi keseluruhan sistem aplikasi Mahoni yang dibuat.

3.1 Pembagian Metode Penyelesaian Masalah

Sesuai dengan latar belakang dan pertanyaan penelitian yang sudah dijelaskan pada bab 1, penelitian ini dapat dibagi menjadi tiga ruang lingkup masalah seperti pada Tabel 3.1. Ruang lingkup masalah yang dibahas dalam penelitian ini adalah pengembangan servis-servis Mahoni, arsitektur *event-driven*, dan arsitektur *big data*. Setiap ruang lingkup memiliki penjelasan metode penelitian, implementasi, dan evaluasi yang berbeda-beda dan dibahas dalam babnya masing-masing.

Tabel 3.1 Pembagian Ruang Lingkup

Identifikasi Masalah	Sub Masalah	Ruang Lingkup Masalah	Bab Penyelesaian Masalah
Fitur-fitur aplikasi	1. Merancang fitur yang sesuai dengan kebutuhan pengguna.	Pengembangan servis-servis	Bab 4
Mahoni	2. Mengimplementasi servis sesuai dengan fitur yang dapat dibuat.	Mahoni	
Merancang arsitektur <i>event-driven</i>	1. Membangun arsitektur <i>loosely-coupled</i> . 2. Membangun sistem <i>event-driven</i> dengan <i>throughput</i> yang tinggi.	Arsitektur <i>event-driven</i>	Bab 5
Merancang arsitektur <i>big data</i>	1. Membangun layer-layer arsitektur dengan <i>framework</i> yang tepat. 2. Membangun <i>dashboard</i> sebagai <i>output</i> . 3. Membangun arsitektur yang <i>scalable</i> dan <i>reliable</i> .	Arsitektur <i>big data</i>	Bab 6

Pengembangan servis-servis Mahoni ditentukan dari daftar fitur aplikasi Mahoni yang dibuat berdasarkan berbagai kebutuhan pengguna. Kebutuhan tersebut dicari dengan menggunakan metode tertentu kemudian menjadi masukan untuk pengembangan servis yang mendukung jalannya fitur pada aplikasi Mahoni. Tiap servis dikembangkan melalui beberapa tahapan untuk menghasilkan produk yang sesuai dengan kebutuhan. Hal ini dibahas lebih detail pada bab 4 terkait pengembangan servis-servis Mahoni.

Agar setiap servis dapat berkomunikasi dengan baik, penelitian ini menggunakan pendekatan arsitektur *event-driven*. Arsitektur ini memiliki tantangan tersendiri di mana setiap komponennya harus memiliki sifat *loosely-coupled* dan dapat menghasilkan *throughput* yang tinggi. Sifat *loosely-coupled* ini dapat membantu aplikasi Mahoni ke depannya apabila akan dilakukan penambahan servis ataupun komponen IoT baru karena proses pengembangannya bisa terisolasi dari komponen yang sudah ada sebelumnya. Arsitektur yang dibuat juga harus bisa mengakomodasi *throughput* yang tinggi akibat data-data yang masuk melalui komponen IoT dan *request* dari pengguna aplikasi Mahoni. Pembahasan lebih detail mengenai hal ini terdapat pada bab 5 terkait arsitektur *event-driven*.

Sumber data yang berasal dari sensor-sensor udara dan aktivitas-aktivitas pengguna memiliki permasalahan yang dapat ditangani dengan implementasi *big data*, yaitu jumlah data yang besar serta kecepatan data yang mendekati *real-time*. Implementasi *big data* menghasilkan beberapa keluaran di antaranya adalah *dashboard*. Untuk menghasilkan *dashboard* tersebut dibutuhkan arsitektur *big data* dengan *framework* yang tepat untuk setiap *layer* arsitektur. Arsitektur *big data* yang dibuat harus memiliki sifat *scalable* dan *reliable* agar dapat menghadapi perubahan-perubahan yang terjadi akibat perkembangan jumlah dan kebutuhan pengguna. Pembahasan ini dijelaskan lebih detail pada bab 6 terkait arsitektur *big data*.

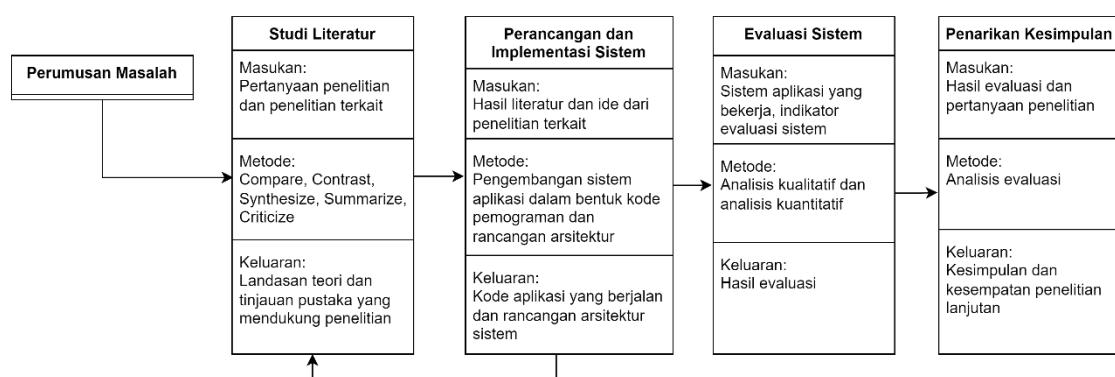
3.2 Metodologi Umum

Metodologi untuk menyelesaikan tiga permasalahan di atas secara umum memiliki pendekatan yang sama. Pendekatan yang dilakukan berupa pendekatan kuantitatif dan kualitatif dalam proses evaluasinya. Namun, masing-masing ruang lingkup masalah memiliki skenario evaluasi yang berbeda. Hasil evaluasi lalu diukur dengan matriks

evaluasi yang telah ditentukan dan dapat dilakukan analisis serta penarikan kesimpulan penelitian.

3.2.1 Tahapan Penelitian

Proses penelitian ini secara umum dibagi menjadi 5 tahapan penelitian yaitu perumusan masalah, studi literatur, perancangan dan implementasi sistem, evaluasi sistem, dan penarikan kesimpulan. Seperti yang dapat dilihat pada Gambar 3.1, masing-masing tahapan memiliki masukan, metode, dan keluaran yang berbeda-beda. Untuk tahapan pertama, setiap ruang lingkup memiliki masukan dan metode tersendiri yang dijelaskan lebih lanjut pada masing-masing bab. Hasil dari tahapan pertama adalah pertanyaan penelitian yang menjadi masukan untuk tahapan berikutnya.



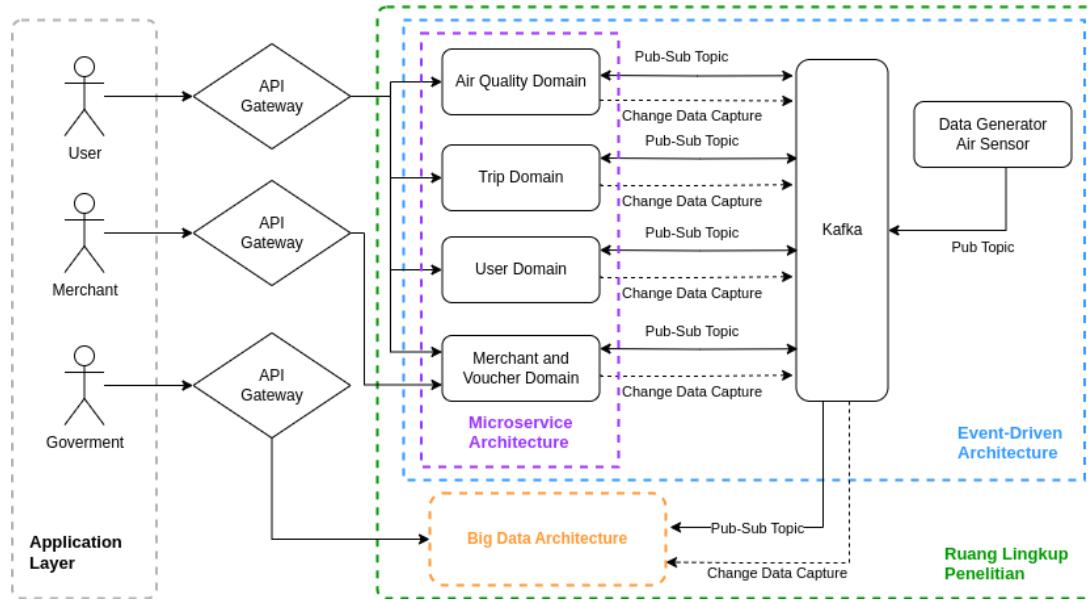
Gambar 3.1 Bagan Tahapan Penelitian Secara Umum

Tahapan kedua penelitian ini yaitu studi literatur secara mendalam berdasarkan pertanyaan penelitian yang telah disusun. Sumber-sumber yang didapatkan baik dari penelitian terdahulu, hasil wawancara, maupun hasil analisis kebutuhan dari masyarakat dan pemerintah juga diolah untuk mendapatkan landasan teori yang mendukung penelitian. Untuk mengolah sumber-sumber tersebut dilakukan 5 teknik yaitu *compare*, *contrast*, *criticize*, *synthesize*, dan *summarize*. Sumber-sumber yang telah didapatkan dikumpulkan dan dicari persamaan serta perbedaannya dengan melakukan *compare* dan *contrast*. Setelah itu, kelebihan dan kelemahan dari informasi yang sudah didapatkan dicari dengan melakukan *criticize*. Kemudian informasi-informasi tersebut digabungkan dan disimpulkan untuk mendapatkan pemahaman yang lebih mendalam terkait informasi yang mendukung untuk menyelesaikan masing-masing permasalahan.

Setelah mendapatkan informasi berupa konsep dan fondasi yang kuat, tahapan berikutnya yaitu perancangan dan implementasi sistem. Hasil dari studi literatur menjadi acuan utama untuk membuat solusi dari masing-masing permasalahan. Sistem yang sudah dibuat lalu dievaluasi dengan skenario yang ditentukan sesuai dengan masing-masing permasalahan. Sistem tersebut dianalisis secara kualitatif dan kuantitatif berdasarkan hasil skenario. Hasil dari evaluasi yang didapatkan menjadi dasar sebagai penarikan kesimpulan untuk tahap selanjutnya. Pada tahap penarikan kesimpulan, hasil analisis yang telah didapatkan ditarik kesimpulannya. Kesimpulan tersebut digunakan untuk menjawab pertanyaan penelitian dan menjadi masukan bagi penelitian berikutnya.

3.2.2 Gambaran Umum

Pada tahapan perancangan dan implementasi sistem, sistem aplikasi Mahoni dirancang dan dibangun dengan mengimplementasi tiga jenis arsitektur seperti Gambar 3.2. Arsitektur ini terbagi menjadi tiga sesuai dengan ruang lingkup masalah yang ada yaitu arsitektur *microservice*, arsitektur *event-driven* dan arsitektur *big data*. Arsitektur tersebut masing-masing dijelaskan dan diuji pada bab 4, 5 dan 6.



Gambar 3.2 High-Level Arsitektur Mahoni

Pada arsitektur tersebut terdapat beberapa sumber data yang berasal dari pengguna maupun dari sensor. Sumber data sensor udara berasal dari generator data sensor udara yang menghasilkan data buatan setiap detik berdasarkan data sensor udara asli yang

dimanipulasi. Data sensor udara tersebut diambil dari Airly¹ menggunakan API yang disediakan. Selain itu, sumber data juga berasal dari pengguna yang melakukan aktivitas di aplikasi Mahoni. Aktivitas pengguna dikelompokkan menjadi beberapa domain yang di dalamnya terdapat servis untuk mengolah dan menyediakan data yang dibutuhkan.

Sumber data yang sudah diolah menghasilkan beberapa keluaran. Keluaran ini dapat diakses oleh pengguna baik pengguna biasa, mitra usaha, maupun dari pihak pemerintah. Pengguna biasa dapat mengakses data dari semua servis, mitra usaha dapat mengakses data dari servis kupon, dan pihak pemerintah dapat mengakses *dashboard* yang dibuat pada arsitektur *big data*.

3.3 Skenario dan Matriks Evaluasi Umum

Keseluruhan arsitektur yang telah dibangun dilakukan evaluasi untuk menilai apakah kinerja sistem aplikasi Mahoni sudah sesuai dengan kebutuhan atau belum. Evaluasi dilakukan dengan menggunakan skenario pengujian *end-to-end*.

3.3.1 Skenario Evaluasi Umum

Pengujian *end-to-end* merupakan evaluasi untuk memastikan bahwa sistem telah bekerja dengan baik dari awal sampai akhir. Dalam pengujian *end-to-end*, pengujian dilakukan dari sudut pandang pengguna yang menggunakan sistem. Semua komponen yang berkaitan di dalam sistem secara langsung diuji kebenaran implementasinya. Tidak hanya implementasi servis, tetapi implementasi arsitektur *event-driven* dan arsitektur *big data* dapat diuji sekaligus dalam menjalankan pengujian *end-to-end*.

Sebelum menjalankan pengujian *end-to-end*, perlu diketahui *thin-thread* yaitu fungsi atau fitur apa yang ingin dilakukan lakukan pengujian. Setiap *thin-thread* dapat diidentifikasi dengan melihat *business case*, percabangan, kondisi input, kondisi keluaran, dan sebagainya. (Tsai et al., 2001). Identifikasi *thin-thread* dari implementasi servis-servis Mahoni dijelaskan lebih detail pada subbab 4.5. *Thin-thread* yang digunakan pada skenario pengujian *end-to-end* ini diturunkan dari proses bisnis Mahoni yang telah

¹ airly.org

ditentukan pada saat Gemastik dan dilakukan modifikasi pada pengembangan servis-servis Mahoni.

Tabel 3.2 Skenario Pengujian *End-to-End*

No.	Skenario <i>Thin-thread</i>
1.	Mitra usaha sukses mendaftarkan akun di Mahoni
2.	Mitra usaha sukses mendaftarkan kupon dan detailnya ke dalam sistem
3.	Pengguna umum sukses mendaftarkan akun di Mahoni
4.	Pengguna umum sukses mendapatkan poin setelah melakukan perjalanan transportasi umum dan melakukan pencatatan di sistem Mahoni
5.	Pengguna umum sukses menukarkan poin yang dimiliki menjadi kupon
6.	Pengguna umum sukses menggunakan kupon ke mitra usaha

Thin-thread dapat saling bergantung sama lain maupun independen dengan *thin-thread* lainnya. Hal tersebut tergantung dengan skenario pengujian yang dibangun atas *thin-thread* yang dipilih. Dalam evaluasi umum pada bagian ini, dibuat skenario pengujian di mana pengguna melakukan perjalanan transportasi umum hingga melakukan penukaran poin menjadi kupon. Skenario pengujian *end-to-end* dapat dilihat pada Tabel 3.2. *Thin-thread* yang menyusun skenario ini bergantung satu sama lain sehingga rangkaian *thin-thread* tersebut dijalankan secara berurutan.

3.3.2 Metrik Evaluasi Umum

Evaluasi umum sistem aplikasi Mahoni dilakukan dengan menggunakan pendekatan kuantitatif dan kualitatif. Evaluasi kuantitatif dijalankan dengan mengetahui banyaknya kesalahan berupa eror yang terjadi pada sistem. Evaluasi tersebut diukur dengan menggunakan metrik *error rate*, yaitu persentase eror yang terjadi dari keseluruhan pengujian yang dilakukan. Selain itu, evaluasi kuantitatif juga dilakukan dengan pencatatan berapa lama waktu yang dibutuhkan oleh pengguna untuk menyelesaikan suatu skenario menggunakan metrik *average response time per second*. Hasil dari evaluasi ini digunakan sebagai evaluasi implementasi arsitektur *event-driven*.

Kemudian, evaluasi kualitatif dijalankan dengan melihat kesesuaian keluaran yang diharapkan dengan keluaran saat melakukan pengujian. Hal tersebut menggambarkan keadaan pengguna saat melakukan berbagai aktivitas pada sistem aplikasi Mahoni.

Evaluasi kualitatif juga dilakukan dengan melihat aliran data yang tercatat di *dashboard*. Selain itu, evaluasi dilakukan dengan mengecek keberadaan data pada *database* dan *data warehouse* arsitektur *big data*. Hasil dari evaluasi ini digunakan sebagai evaluasi pengembangan servis-servis Mahoni serta implementasi arsitektur *big-data*. Hasil dari evaluasi umum dengan pendekatan kuantitatif maupun kuantitatif dijelaskan pada bab 7 dan menjadi bahan pertimbangan untuk pengembangan berikutnya.

BAB 4

PENGEMBANGAN SERVIS-SERVIS MAHONI

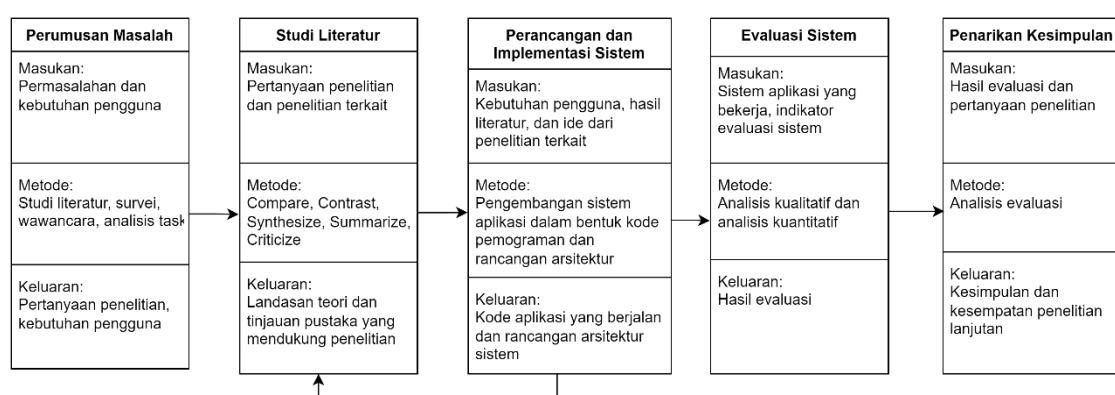
Pengembangan servis-servis Mahoni dilakukan untuk membangun layanan dari fitur-fitur aplikasi Mahoni. Bab ini membahas lebih detail mengenai metodologi serta tahapan-tahapan yang dilakukan untuk pengembangan servis-servis. Tahapan tersebut dimulai dari praperancangan untuk mendapatkan kebutuhan fitur, yang dilanjutkan dengan tahap perancangan, pengembangan, dan evaluasi kepada servis yang dikembangkan.

4.1 Metode dan Metodologi

Pendekatan yang digunakan untuk pekerjaan ini adalah metode penelitian dan pengembangan (*Research and Development*) dengan menggabungkan pendekatan kualitatif dan kuantitatif. Metode ini digunakan untuk mengembangkan produk yang sudah ada atau menciptakan suatu produk baru (Sugiyono & Lestari, 2021). Pekerjaan dilakukan untuk mengembangkan servis dari Mahoni yang didasari atas evaluasi kualitatif proses penelitian dalam *requirement gathering*. Servis-servis Mahoni yang dikembangkan kemudian dievaluasi dengan dilakukan pengujian.

4.1.1 Tahapan Penelitian

Tahapan penelitian dalam pengembangan servis-servis Mahoni sama seperti yang telah dijelaskan di Subbab 3.2.1 yaitu perumusan masalah, studi literatur, perancangan dan implementasi sistem, evaluasi sistem, dan penarikan kesimpulan. Namun, terdapat keterangan lebih lanjut untuk tahapan perumusan masalah seperti pada Gambar 4.1.

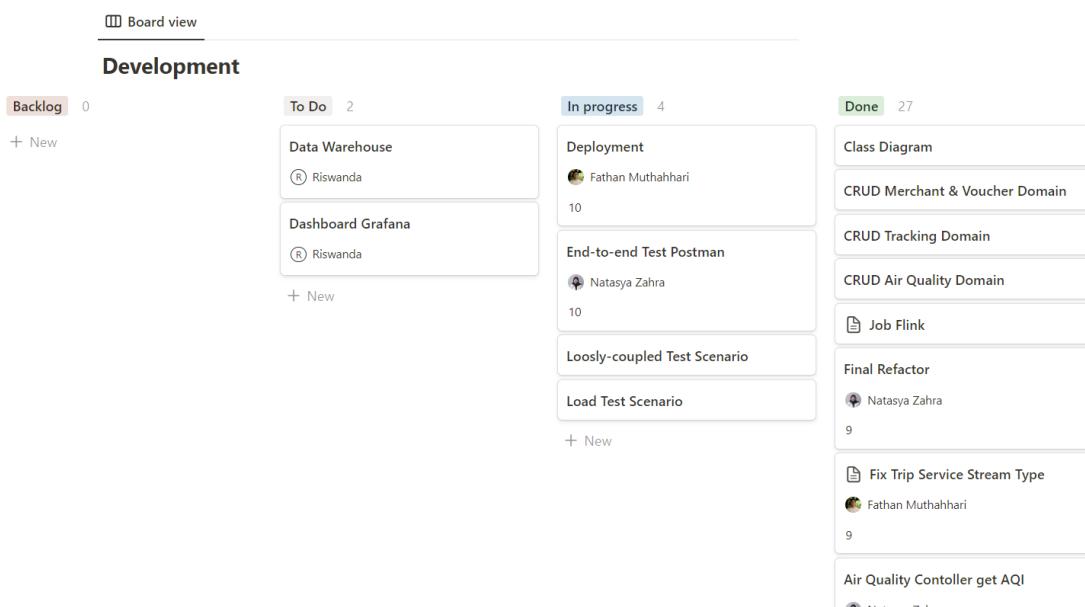


Gambar 4.1 Tahapan Penelitian Pengembangan Servis-Servis Mahoni

Perumusan masalah dilakukan dengan melakukan *requirement gathering* mengikuti langkah-langkah yang diusulkan oleh Silhavy et al. (2011). Metode pengumpulan data yang digunakan dalam *requirement gathering* adalah survei dengan kuesioner, wawancara, dan analisis *task*. Proses dan hasil dari *requirement gathering* dijelaskan pada subbab 4.2. Keluaran dari tahapan ini adalah kebutuhan pengguna yang menjadi masukan untuk tahapan selanjutnya.

4.1.2 Metodologi Pengembangan

Dalam melakukan pengembangan servis-servis Mahoni, digunakan metodologi dalam melakukan pekerjaan dan pengelolaan kerja yaitu Scrumban. Proses pekerjaan dilakukan dengan mengadopsi ritual yang setiap satu siklus dilakukan pada Sprint, yaitu *sprint planning*, *daily stand-up*, dan *retrospective*. Alih-alih menunggu satu siklus Sprint, kegiatan tersebut dilakukan setiap minggunya secara kasual. Untuk mendukung dokumentasi dan manajemen *task* yang akan, sedang, dan sudah dilakukan, papan Scrumban digunakan selama proses pengembangan. Cuplikan papan tersebut dapat dilihat pada Gambar 4.2.



Gambar 4.2 Papan Scrumban

Pada papan Scrumban tersebut, terdapat empat buah bagian kategori *task*, yaitu: (1) *Backlog* untuk mendaftarkan semua pekerjaan secara abstrak, (2) *Todo* untuk

mendaftarkan *task* yang akan dikerjakan pada suatu minggu, (3) *In progress* untuk meletakkan visualisasi *task* apa saja yang sedang dikerjakan pada suatu minggu, yang mana jumlah maksimal *task* untuk setiap orang adalah dua *task*, dan (4) *Done* untuk mendaftarkan semua *task* yang telah selesai.

4.1.3 Skenario Pengujian

Beberapa pengujian dilakukan untuk pengembangan servis-servis Mahoni. Pengujian tersebut dibagi menjadi tiga bagian secara *bottom-up*, yaitu pengujian kualitas dan kebenaran kode, pengujian integrasi dependensi, dan pengujian kesesuaian proses bisnis. Pengujian tersebut didasarkan dari panduan pengujian *microservices* oleh Clemson (2014) yang menggunakan konsep piramida pengujian. Ikhtisar singkat mengenai skenario pengujian terdapat pada Tabel 4.1.

Tabel 4.1 Skenario Pengujian Pengembangan Servis-Servis Mahoni

No	Pengujian	Tujuan	Instrumen
1	<i>Unit Testing</i>	Menguji apakah kode yang dibuat sesuai dengan tujuannya	Skrip <i>unit test</i> dan servis yang telah dibuat
2	<i>Integration Testing</i>	Menguji apakah servis yang telah dibuat dapat terhubung dengan dependensi yang diperlukannya	Skrip <i>integration test, message broker</i> , dan <i>database</i>
3	<i>End-to-End Testing</i>	Menguji apakah proses bisnis dapat berjalan secara semestinya di servis yang telah dibuat	Alat <i>end-to-end testing</i> dan servis yang telah di-deploy

Pengujian kebenaran implementasi dan kualitas kode dilakukan dengan menggunakan *unit test*. Unit testing merupakan pengujian implementasi untuk setiap komponen kecil pada kode, misalnya *method* ataupun *function*. *Unit test* umumnya dibuat terlebih dahulu sebelum menulis kode agar memastikan bahwa kode yang dibuat benar dan sesuai dengan *test* yang dibuat. Kode pada setiap servis yang dievaluasi dengan *unit test* yaitu kode yang merepresentasikan proses bisnis ataupun komponen tertentu dari servis tersebut.

Setelah implementasi setiap servis diuji dengan menggunakan *unit test*, langkah yang dilanjutkan selanjutnya adalah dengan melakukan *integration testing*. *Integration testing* dilakukan untuk menguji keterhubungan servis dengan dependensi yang digunakannya, dalam hal ini yaitu *database* untuk penyimpanan data dan *message broker* untuk aliran

pesan antar servis dalam menunjang implementasi arsitektur *event-driven*. Pendekatan pengujian keterhubungan dependensi yang dipakai adalah dengan menyisipkan kegiatan unit testing dependensi tersebut. Apabila *unit testing*-nya berhasil, maka secara langsung dapat diketahui bahwa integrasi dependensi juga berhasil.

Pengujian terakhir yang dilakukan adalah *end-to-end testing* yang bertujuan untuk menguji apakah implementasi servis secara keseluruhan sesuai dengan proses bisnis yang telah ditetapkan sebelumnya. Penjelasan umum dan skenario untuk pengujian *end-to-end* tertera pada subbab 3.3. Telah disebutkan bahwa dalam pengujian *end-to-end* perlu diketahui *thin-thread* yang menyusun skenario tersebut. Pada pekerjaan ini, dilakukan identifikasi seluruh *thin-thread* dengan membuat peta *thin-thread* berdasarkan seluruh *endpoint* setiap servis yang dibuat. Kemudian, setiap *thin-thread* tersebut dijalankan untuk memastikan proses bisnis berjalan secara semestinya.

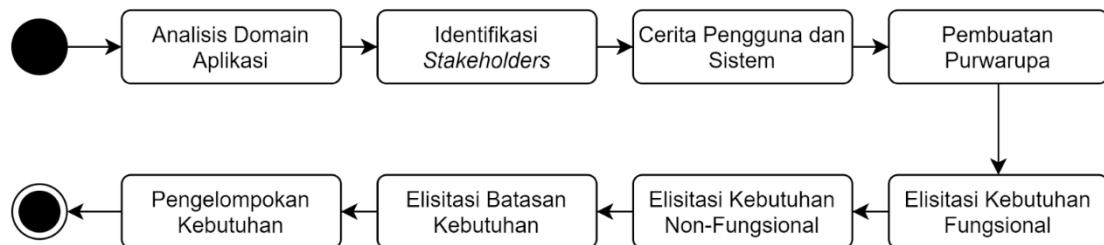
4.1.4 Matriks Evaluasi

Masing-masing tahapan dalam skenario pengujian servis-servis Mahoni memiliki matriks evaluasinya tersendiri. Secara singkat, hal tersebut telah disampaikan pada subbab sebelumnya. Terdapat dua pendekatan evaluasi yang dilakukan, yaitu evaluasi kualitatif dan kuantitatif. Evaluasi kualitatif dilakukan pada semua tahap yaitu *unit testing*, *integration testing*, dan *end-to-end testing* dengan melihat apakah setiap pengujian berhasil dilakukan. Dengan melihat keberhasilan pengujian, hal tersebut mengimplikasikan pengembangan yang dilakukan sesuai dengan dokumentasi proses bisnis yang dibuat serta tujuan yang ditentukan tercapai.

Kemudian, evaluasi kuantitatif dilakukan pada tahap unit testing, yaitu dengan melihat cakupan kode yang dikenakan pengujian dengan menggunakan matriks *code coverage*. Besaran nilai *code coverage* cukup beragam tergantung dari kebutuhan dan standar pengembang. Namun, berdasarkan kesimpulan diskusi yang dilakukan oleh pengembang di forum StackOverflow (*What Is a Reasonable Code Coverage % for Unit Tests (and Why)?*, n.d.) dan artikel oleh Atlassian (Sten Pittet, n.d.), nilai tersebut umumnya berada dari kisaran 80%. Untuk pengembangan ini, *code coverage* minimal yang dipakai adalah 85% karena terdapat cukup banyak berkas konfigurasi yang mendukung pekerjaan ini.

4.2 Praperancangan Servis

Pada tahapan praperancangan servis, dilakukan pengumpulan kebutuhan-kebutuhan yang akan diimplementasikan di sistem dengan *requirement gathering*. Tahapan ini sangat krusial demi menciptakan sistem yang sesuai dengan kebutuhan pengguna.



Gambar 4.3 Proses *Requirement Gathering*

Sumber: Silhavy et al. (2011), telah diolah kembali

Terdapat beberapa tahapan dalam melakukan *requirement gathering* menurut Silhavy et al. (2011) dimulai dari analisis domain aplikasi hingga pengelompokan kebutuhan. Tahapan-tahapan tersebut dapat dilihat pada Gambar 4.3 dan penjelasan lanjut tiap tahapannya dijelaskan pada subbab.

4.2.1 Analisis Domain Aplikasi

Tahap pertama dari proses melakukan *requirement gathering* adalah analisis domain aplikasi. Analisis domain aplikasi merupakan upaya untuk mengetahui dan memahami karakteristik, kondisi, dan batasan dari aplikasi yang ingin dibuat. Proses ini melibatkan pengumpulan informasi tidak hanya seputar domain pengembangan aplikasi, tetapi juga domain tema atau topik yang menjadi fokus aplikasi. Untuk memulai proses analisis, penting untuk memahami tujuan dari aplikasi. Setelah itu, langkah yang dapat dilakukan selanjutnya adalah menelaah informasi yang berkaitan dengan domain.

Aplikasi Mahoni bertujuan sebagai upaya untuk mendukung penggunaan transportasi umum dengan insentif pemberian poin yang kuantitasnya relatif terhadap tingkat kualitas udara. Oleh karena itu, domain tema aplikasi yang diangkat Mahoni menggabungkan konsep kualitas udara, transportasi umum, dan transaksi poin. Dari ketiga konsep tersebut, dipelajari terminologi, proses, dan informasi terkait lainnya. Informasi yang

didapatkan secara umum tercantum pada bab 2. Kemudian, domain pengembangan aplikasi Mahoni adalah aplikasi berbasis *mobile* dan web. Namun, pada pekerjaan ini hanya dilakukan pengembangan servis yang merupakan aplikasi web yang kemudian dijalankan di *cloud*.

4.2.2 Identifikasi Stakeholders

Tahapan krusial berikutnya adalah identifikasi *stakeholders*. Tujuan dari identifikasi *stakeholders* yaitu untuk mengetahui pihak berkepentingan yang terlibat dengan aplikasi yang ingin dibuat. Mereka memiliki tujuan masing-masing yang ingin dicapai dan dapat memberi masukan yang berperan dalam pembentukan proses bisnis. Penentuan pihak tersebut dilakukan setelah memahami analisis domain secara mendalam dengan melihat siapa pihak yang terlibat untuk setiap domain.

Terdapat tiga pihak berkepentingan yang terlibat dengan Mahoni, yaitu pengguna umum, mitra usaha, dan pengelola daerah. Ketiga pihak tersebut secara bersama-sama terhubung satu sama lain di dalam sistem Mahoni, dengan mitra usaha berperan dalam transaksi poin dan pengguna umum serta pengelola daerah berperan dalam semua domain tema aplikasi. Penjelasan detail mengenai ketiga pihak tersebut dijelaskan pada subbab berikutnya.

4.2.3 Cerita Pengguna dan Sistem

Setelah mengetahui pihak yang terlibat dalam aplikasi Mahoni yaitu pengguna umum, mitra usaha, dan pengelola daerah, tahap selanjutnya yang dilakukan adalah mengidentifikasi peran dari setiap pihak. Peran masing-masing pihak dapat didokumentasikan dengan menggunakan cerita pengguna (*user stories*). Cerita pengguna dapat memuat cerita sistem (*system stories*) yang berinteraksi langsung dengan pengguna. Tujuan dari pembuatan cerita pengguna ini adalah sebagai jembatan pemahaman antara pihak yang terlibat dengan pengembang aplikasi. Cerita pengguna ini tidak menceritakan secara detail informasi teknis dalam pengembangan aplikasi tetapi memiliki cukup untuk pemetaan kebutuhan dan pembuatan *use case* pada tahap *requirement gathering*.

Pengguna umum merupakan pengguna utama dari aplikasi Mahoni. Pengguna ini bisa mendapatkan poin setelah menaiki transportasi umum. Kuantitas poin ditentukan oleh kualitas udara dari daerah yang dilewatinya. Poin yang telah dikumpulkan dapat

ditukarkan menjadi kupon yang tersedia. Selain itu, pengguna ini dapat melihat informasi umum mengenai kualitas udara. Pengguna umum nantinya menggunakan aplikasi Mahoni berbasis *mobile*. Berikut ini merupakan cerita pengguna untuk pengguna umum:

1. Sebagai pengguna umum, saya ingin dapat melihat informasi umum mengenai kualitas udara dalam sistem secara langsung agar kebutuhan untuk mengetahui informasi tersebut terpenuhi.
2. Sebagai pengguna umum, saya ingin perjalanan transportasi umum saya tercatat dalam sistem agar saya mendapatkan poin.
3. Sebagai pengguna umum, saya ingin dapat menukarkan poin yang saya miliki menjadi kupon dalam sistem agar saya bisa mendapatkan keuntungan dari pemakaian kupon tersebut.

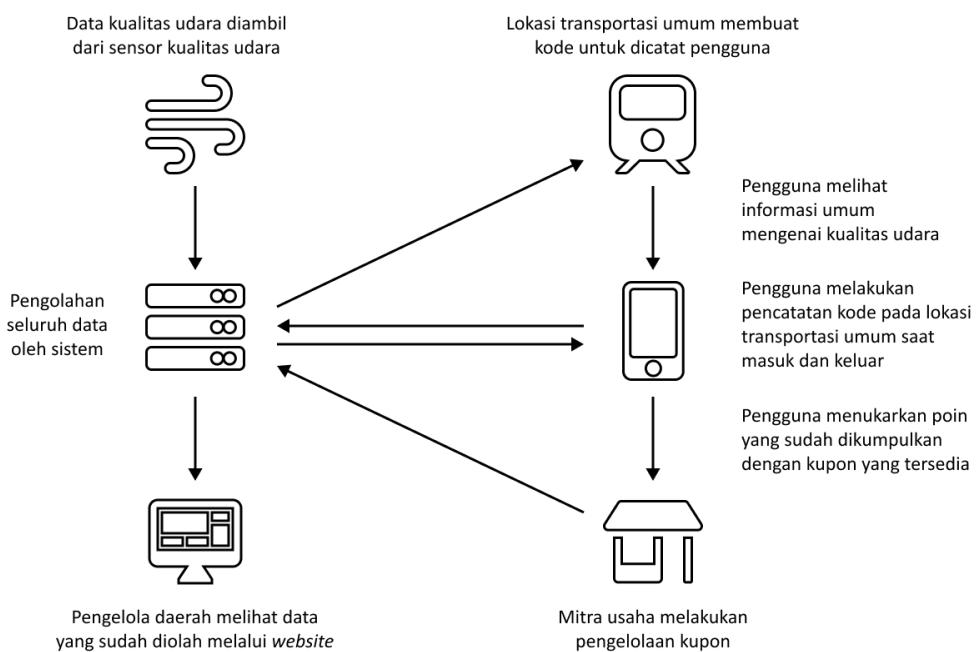
Mitra usaha merupakan pihak yang bertanggung jawab atas pengelolaan kupon pada aplikasi Mahoni. Pengguna ini menyediakan kupon yang nantinya tersedia di sistem Mahoni untuk pengguna umum dapat tukarkan dengan poin yang mereka peroleh. Mitra usaha yang terdaftar adalah pihak ataupun badan usaha yang telah disetujui oleh pengelola daerah. Pengguna ini nantinya menggunakan aplikasi Mahoni berbasis web. Berikut ini merupakan cerita pengguna untuk mitra usaha:

1. Sebagai mitra usaha, saya ingin dapat mengelola kupon pada sistem sehingga penjualan usaha saya meningkat.

Pengelola daerah merupakan pihak yang mengelola daerah di mana basis aplikasi kota cerdas Mahoni dijalankan. Pengelola daerah juga merupakan pihak yang bertanggung jawab atas transportasi umum yang terintegrasi dengan Mahoni dan mitra usaha yang beroperasi di daerah tersebut. Pengguna ini dapat memantau data kualitas udara, penggunaan transportasi umum, dan transaksi kupon secara umum dari aplikasi Mahoni. Pengelola daerah nantinya menggunakan aplikasi Mahoni berbasis web. Berikut ini merupakan cerita pengguna untuk pengelola daerah:

1. Sebagai pengelola daerah, saya ingin dapat mendapatkan informasi mengenai data kualitas udara, penggunaan transportasi umum, dan mitra usaha agar informasi tersebut dapat saya gunakan untuk keperluan pengelolaan daerah.

Pengelola daerah memiliki kebutuhan terhadap data kualitas udara, penggunaan transportasi umum, dan mitra usaha yang cukup besar. Oleh karena itu, implementasi kebutuhan tersebut tidak dilakukan pada pekerjaan ini dengan membuat servis, tetapi dilakukan oleh pekerjaan di bab 6 dengan menggunakan arsitektur *big data*. Cerita pengguna untuk pengelola daerah secara lengkap terdapat pada subbab 6.2.2.



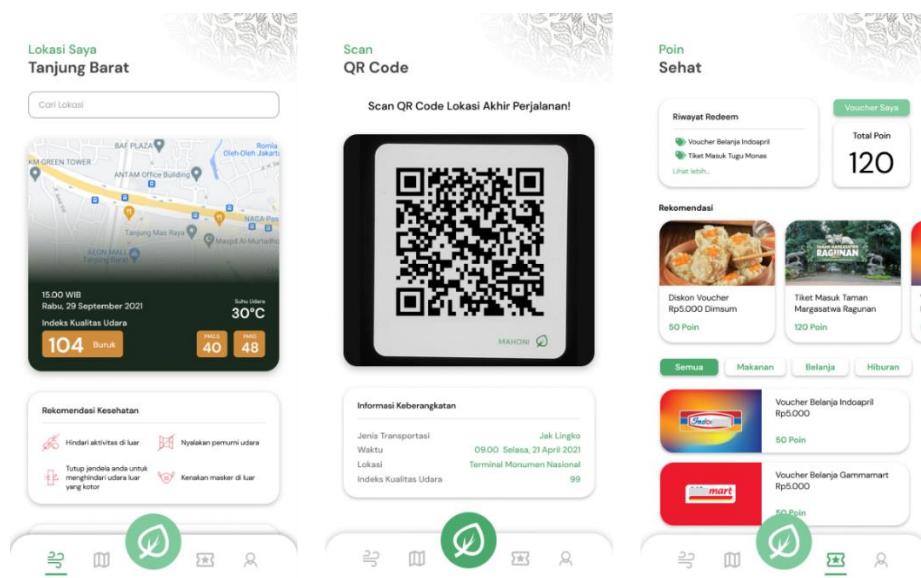
Gambar 4.4 Proses Bisnis Mahoni

Dari cerita pengguna yang telah ditentukan, penggambaran proses bisnis tahap awal dapat dilakukan. Gambar 4.4 menunjukkan alur interaksi antar pengguna Mahoni baik pengguna umum, mitra usaha, maupun pengelola daerah. Proses bisnis ini menjadi masukan untuk tahapan selanjutnya yaitu pembuatan purwarupa dan elisitasi kebutuhan. Perbaikan dan finalisasi proses bisnis dilakukan setelah mendapatkan kebutuhan pengguna yang lebih detail setelah melakukan elisitasi kebutuhan. Diagram proses bisnis tersebut beserta penjelasannya dapat dilihat pada subbab 4.3.

4.2.4 Pembuatan Purwarupa

Cerita pengguna yang telah dibuat pada tahapan sebelumnya dikembangkan lebih lanjut menjadi purwarupa fidelitas tinggi. Pembuatan purwarupa fidelitas tinggi ditujukan untuk menggambarkan bagaimana nantinya aplikasi ini dirancang. Proses desain dan pembuatan purwarupa telah dilakukan pada saat Gemastik. Metode dalam proses desain yang digunakan adalah pendekatan *genius design*, yaitu perancangan desain berdasarkan asumsi dan pengetahuan perancang serta tidak ada campur tangan dari pengguna. Intipan beberapa desain halaman aplikasi Mahoni dapat dilihat pada Gambar 4.5.

Dalam hasil desain tersebut dapat dilihat tiga halaman yang merupakan halaman utama untuk masing-masing konsep domain aplikasi yang dibawakan Mahoni. Halaman pertama (kiri) menunjukkan halaman utama untuk domain pemantauan kualitas udara. Pada halaman ini terdapat informasi mengenai kualitas udara, rekomendasi kesehatan, dan beberapa fitur penunjang lain yang salah satunya adalah fitur pencarian lokasi. Kemudian, halaman kedua (tengah) menunjukkan halaman utama untuk domain pencatatan penggunaan transportasi umum yang dilakukan dengan melakukan pindai QR *Code*. Selain itu terdapat juga informasi mengenai perjalanan transportasi umum pengguna. Terakhir, halaman ketiga (kanan) menunjukkan halaman utama untuk domain penukaran poin menjadi kupon. Halaman ini berisikan informasi terkait jumlah poin dan daftar kupon yang tersedia.



Gambar 4.5 Purwarupa Fidelitas Tinggi Aplikasi Mahoni

Fitur-fitur yang tertera di rancangan desain ini mengacu kepada *requirement gathering* yang telah dilakukan pada saat Gemastik. Walaupun demikian, hasil *requirement gathering* yang dilakukan hanya berdasarkan observasi yang tidak melibatkan pengguna dan terdapat kemungkinan kurang sesuai dengan kebutuhan saat ini. Oleh karena itu, hasil dari perancangan purwarupa fidelitas tinggi digunakan sebagai alat bantu visualisasi dalam tahap *requirement gathering* selanjutnya yaitu elisitasi kebutuhan.

4.2.5 Elisitasi Kebutuhan Fungsional

Kebutuhan fungsional mendeskripsikan fungsi apa saja yang dibutuhkan oleh suatu pengguna dan bagaimana mekanisme fungsi tersebut bekerja. Dalam tahapan ini, dilakukan pengumpulan kebutuhan fungsional untuk mengetahui kebutuhan pengguna terhadap fungsionalitas sistem. Metode yang digunakan untuk mengumpulkan kebutuhan fungsional adalah analisis *task* dan wawancara. Metode tersebut dipilih untuk menyeimbangkan kebutuhan fungsionalitas yang dibutuhkan pengguna dan ditawarkan oleh aplikasi dengan tema serupa.

4.2.5.1 Analisis Task

Analisis *task* dilakukan dengan melakukan observasi fitur dan informasi apa saja yang tersedia di aplikasi pemantau kualitas udara, pendamping transportasi, dan aplikasi yang menyediakan sistem penukaran poin menjadi kupon. Aplikasi yang dipilih memiliki peringkat di atas 4.0 dan telah diunduh lebih dari 50 ribu kali pada Google Play Store². Tahapan ini sebelumnya telah dilakukan saat Gemastik tetapi dilakukan kembali karena pembaruan aplikasi seiring waktu. Luaran dari proses ini adalah daftar perbandingan fitur yang dapat diimplementasikan oleh Mahoni.

Seperti yang dapat dilihat pada Tabel 4.2, terdapat lima aplikasi pemantau kualitas udara yang dipilih untuk perbandingan, yaitu (1) Nafas³, (2) IQAir⁴, (3) PlumeLabs⁵, (4) BreezoMeter⁶, dan (5) Weather.com⁷. Hasil dari observasi menunjukkan terdapat delapan fitur utama pada aplikasi tersebut, dengan: fitur informasi kualitas udara dan pencarian

² <https://play.google.com/>

³ <https://play.google.com/store/apps/details?id=id.co.nafas.android>

⁴ <https://play.google.com/store/apps/details?id=com.airvisual>

⁵ <https://play.google.com/store/apps/details?id=com.plumelabs.air>

⁶ <https://play.google.com/store/apps/details?id=app.breezometer>

⁷ <https://weather.com/>

kualitas udara diimplementasikan oleh semua aplikasi; fitur riwayat kualitas udara, prediksi kualitas udara, rekomendasi kegiatan, dan visualisasi peta kualitas udara diimplementasi oleh sebagian besar aplikasi; fitur artikel dan *push-notification* hanya diimplementasi oleh sedikit aplikasi saja.

Tabel 4.2 Hasil Analisis Task Aplikasi Kualitas Udara: Kualitas Udara

Kategori	No	Fitur	(1)	(2)	(3)	(4)	(5)
Kualitas Udara	A1	Informasi Kualitas Udara	✓	✓	✓	✓	✓
	A2	Pencarian Kualitas Udara	✓	✓	✓	✓	✓
	A3	Riwayat Kualitas Udara	✓	✓	✓		
	A4	Prediksi Kualitas Udara		✓	✓		✓
	A5	Rekomendasi Kegiatan	✓	✓	✓	✓	
	A6	Visualisasi Peta Kualitas Udara	✓	✓	✓	✓	
	A7	Artikel		✓			
	A8	<i>Push-notification</i>	✓		✓		

Tabel 4.3 Hasil Analisis Task Aplikasi Kualitas Udara: Transportasi dan Poin

Kategori	No	Fitur	(1)	(2)	(3)	(4)	(5)	(6)
Transportasi	B1	Informasi Perjalanan	✓	✓	✓	✓		
	B2	Riwayat Perjalanan	✓	✓				
	B3	Melihat Jadwal Perjalanan			✓	✓		
	B4	Melihat Rute Perjalanan	✓	✓	✓	✓		
	B5	Membeli Tiket Perjalanan	✓		✓			
Poin dan Kupon	C1	Informasi Poin	✓	✓	✓		✓	✓
	C2	Daftar Kupon Tersedia	✓	✓			✓	✓
	C3	Detail Kupon Tersedia	✓	✓			✓	✓
	C4	Daftar Kupon Dimiliki	✓				✓	✓
	C5	Detail Kupon Dimiliki		✓			✓	✓
	C6	Kategorisasi Pengguna	✓				✓	✓
	C7	Kategorisasi Kupon	✓	✓			✓	✓
	C8	<i>Merchant</i> Terdekat					✓	✓

Seperti yang dapat dilihat pada Tabel 4.2, terdapat enam aplikasi pendamping transportasi ataupun yang menerapkan sistem poin dan kupon yang dipilih untuk perbandingan, yaitu (1) Gojek⁸, (2) Grab⁹, (3) MRT-J¹⁰, (4) C-Access¹¹, (5) MyTelkomsel¹², dan (6) Alfagift¹³. Hasil dari observasi menunjukkan terdapat lima fitur utama pada aplikasi pendamping transportasi, dengan: fitur informasi perjalanan dan melihat rute perjalanan diimplementasikan oleh semua aplikasi; fitur riwayat perjalanan, melihat jadwal perjalanan, dan membeli tiket diimplementasi oleh beberapa aplikasi. Kemudian, terdapat delapan fitur utama terkait sistem poin dan kupon pada aplikasi dengan: fitur informasi poin, daftar kupon tersedia, detail kupon tersedia, dan kategorisasi kupon diimplementasi oleh sebagian besar aplikasi; fitur daftar kupon dimiliki, detail kupon dimiliki, dan kategorisasi pengguna diimplementasi oleh beberapa aplikasi; fitur *merchant* terdekat diimplementasi oleh satu aplikasi saja.

Dari seluruh fitur utama yang diidentifikasi terdapat pada aplikasi yang dipilih, tidak semua fitur tersebut diimplementasikan Mahoni karena perbedaan proses bisnis dan kebutuhan yang mendasari pembuatan fitur. Namun, beberapa fitur tersebut dipertimbangkan sebagai kandidat untuk diimplementasikan berdasarkan kesesuaianya dengan proses bisnis mahoni, kemungkinannya untuk dapat diimplementasikan, dan temuan bahwa fitur tersebut diimplementasikan oleh banyak aplikasi. Seluruh fitur kualitas udara kecuali visualisasi peta kualitas udara dipilih sebagai kandidat fitur yang diimplementasikan pada halaman kualitas udara mahoni karena keterbatasan data dan tidak ada penentuan batasan daerah atau kota sebagai studi kasus kota cerdas pada pekerjaan ini. Lalu, kandidat fitur transportasi adalah fitur informasi perjalanan, melihat rute perjalanan dan riwayat perjalanan, serta kandidat fitur poin dan kupon adalah seluruh fitur kecuali *merchant* terdekat.

4.2.5.2 Wawancara

Tahapan wawancara diperuntukkan mendapatkan pandangan yang lebih mendalam akan pengalaman dan kebutuhan partisipan dalam menggunakan layanan yang serupa dengan

⁸ <https://play.google.com/store/apps/details?id=com.gojek.app>

⁹ <https://play.google.com/store/apps/details?id=com.grabtaxi.passenger>

¹⁰ <https://play.google.com/store/apps/details?id=com.mrt.jakarta>

¹¹ <https://play.google.com/store/apps/details?id=com.kci.access>

¹² <https://play.google.com/store/apps/details?id=com.telkomsel.telkomselcm>

¹³ <https://play.google.com/store/apps/details?id=com.alfamart.alfagift>

Mahoni, serta pendapat mereka mengenai Mahoni. Wawancara dilakukan secara semi-terstruktur, yang mana menggunakan susunan pertanyaan terstruktur juga pertanyaan eksplorasi untuk lebih dalam menggali ide atau jawaban dari partisipan. Terdapat tiga tema yang dibawakan dalam wawancara, yaitu kualitas udara, penggunaan transportasi umum, dan pengalaman penggunaan sistem tukar poin menjadi kupon.

Untuk masing-masing tema, wawancara dimulai dengan bagian latar belakang partisipan yang kemudian dilanjut dengan pembahasan isu dari tema yang diangkat, pengalaman penggunaan aplikasi sejenis, penggunaan fitur pada aplikasi sejenis, dan eksplorasi ide dari partisipan. Bagian latar belakang partisipan dan pembahasan isu dari tema yang diangkat ditujukan untuk mengetahui pemahaman partisipan secara personal mengenai permasalahan yang menjadi dasar dari tema tersebut. Bagian pengalaman aplikasi dan penggunaan fitur pada aplikasi sejenis menggali informasi mengenai alasan, pengalaman, kebiasaan, dan kebutuhan pengguna dalam menggunakan fitur aplikasi serupa. Bagian eksplorasi ide partisipan merupakan bagian terakhir dan penutup wawancara untuk mengelaborasi kembali ide maupun pertanyaan yang disampaikan partisipan.

Partisipan wawancara dalam keseluruhan penelitian ini dibagi menjadi dua segmen, yaitu pengguna potensial dan ahli. Partisipan pengguna potensial yaitu partisipan yang pernah menggunakan layanan pemantau kualitas udara, transportasi umum, atau sistem poin kupon, sedangkan wawancara kepada ahli dilakukan kepada narasumber yang berasal dari lembaga atau instansi pemerintahan terkait. Wawancara yang dibahas pada pekerjaan ini dilakukan kepada pengguna potensial sedangkan wawancara kepada narasumber ahli dilakukan dan dibahas lebih lanjut pada subbab 6.2.1. Wawancara dilakukan kepada 12 partisipan pengguna potensial yang secara beririsan membahas tema kualitas udara kepada 5 partisipan, transportasi umum kepada 10 partisipan, dan sistem poin dan kupon kepada 8 partisipan. Adanya perbedaan jumlah partisipan pada setiap tema khususnya partisipan tema penggunaan transportasi umum yang tinggi karena penggunaan aplikasi pendamping transportasi adalah hal yang umum, terlebih seluruh partisipan merupakan pengguna potensial yang berdomisili di Jabodetabek. Salah satu hasil wawancara dari partisipan tersebut terdapat pada Lampiran 2.

Tabel 4.4 Kebutuhan Fungsional

Tema	Kebutuhan	Solusi Fitur
Kualitas udara	Mengetahui informasi kualitas udara	Informasi kualitas udara, pencarian kualitas udara
Transportasi	Mengetahui informasi terkait perjalanan transportasi	Informasi perjalanan transportasi
Poin dan kupon	Menukarkan poin menjadi kupon	Informasi poin, daftar kupon, informasi kupon

Kesimpulan hasil dari wawancara yang dilakukan berkenaan dengan kebutuhan fungsional untuk setiap tema dapat dilihat secara singkat pada Tabel 4.4. Untuk tema kualitas udara, partisipan mengetahui permasalahan terkait kualitas udara yaitu polusi udara. Selain itu, partisipan juga menyampaikan permasalahan lain terkait kondisi udara yaitu temperatur udara. Hal tersebut membawa kekhawatiran pengguna akan kondisi udara dan menimbulkan kebutuhan untuk mengetahui kondisi udara. Fitur dari penggunaan aplikasi lain oleh partisipan yang memenuhi kebutuhan ini adalah fitur informasi kualitas udara dan pencarian kualitas udara. Informasi mengenai kondisi maupun kualitas udara yang dibutuhkan pengguna di antaranya adalah AQI, kategori AQI, PM2.5, temperatur, dan kelembaban.

Untuk tema penggunaan transportasi umum, partisipan mengetahui permasalahan terkait transportasi yaitu polusi udara dan kemacetan karena jumlah kendaraan yang tinggi. Permasalahan tersebut melatarbelakangi penggunaan transportasi umum oleh partisipan, di samping dari kemudahan yang ditawarkannya. Partisipan menggunakan aplikasi pendamping transportasi umum untuk memenuhi kebutuhan mereka dalam memesan transportasi dan mengetahui informasi mengenai transportasi umum yang digunakannya seperti jadwal ataupun rute perjalanan. Fitur pemesanan transportasi untuk memenuhi kebutuhan tersebut tidak dapat diimplementasikan pada pekerjaan ini karena tidak termasuk ke dalam proses bisnis Mahoni. Oleh karena itu, kandidat fitur yang dapat diimplementasikan adalah fitur informasi perjalanan transportasi seperti waktu dan tempat perjalanan, serta jenis transportasi yang digunakan.

Untuk tema penggunaan sistem poin dan kupon, tidak terdapat permasalahan yang mendasari penggunaan sistem ini oleh partisipan. Namun, terdapat alasan yang mendasari

penggunaan sistem tersebut oleh partisipan, yaitu keuntungan yang ditawarkan oleh sistem berupa *reward* maupun bonus yang mereka dapatkan dengan melakukan sesuatu seperti berbelanja. Keuntungan tersebut didapatkan dengan menukarkan poin sebagai *reward* penggerjaan *task*. Oleh karena itu, partisipan membutuhkan fitur yang dapat menukarkan poinnya menjadi kupon. Informasi yang dibutuhkan partisipan dalam menggunakan fitur tersebut adalah jumlah poin, daftar kupon, dan informasi detail mengenai kupon seperti nama, deskripsi, syarat dan ketentuan, dan masa berlaku.

4.2.6 Elisitasi Kebutuhan Non-Fungsional

Kebutuhan non-fungsional mendeskripsikan bagaimana sifat dalam mengerjakan kebutuhan fungsional. Beberapa atribut atau sifat yang menggambarkan kebutuhan non-fungsional di antaranya adalah *security*, *scalability*, *usability*, *reliability*, *maintainability*, *performance*, dan *efficiency*. Metode yang digunakan untuk mengumpulkan kebutuhan non-fungsional pada pekerjaan ini adalah wawancara. Kegiatan wawancara telah dilakukan pada subbab 4.2.5.2, tetapi subbab ini memaparkan hasil dari wawancara tersebut berkenaan dengan kebutuhan non-fungsional partisipan.

Dari wawancara yang dilakukan, partisipan tidak begitu memaparkan kebutuhan non-fungsional secara spesifik. Untuk masing-masing tema, kebanyakan partisipan menyampaikan kebutuhan yang sama, yaitu *usability* dalam menggunakan *interface* aplikasi. Keterbacaan, kerapian, ramah pengguna, intuisi, dan aksesibilitas merupakan beberapa aspek *usability* yang dibutuhkan partisipan. Di samping *usability* pada *interface* aplikasi, terdapat partisipan yang memiliki kebutuhan *usability* sistem yaitu akses aplikasi tanpa internet. Kebutuhan-kebutuhan tersebut dapat diterapkan dengan pengembangan *interface* yang memenuhi *usability goals*, ataupun dengan penerapan *cache* di *client* pengguna. Namun, hal tersebut bukan merupakan fokus dari pekerjaan ini sehingga pemenuhan kebutuhan tersebut tidak dilakukan.

Kendati minimnya kebutuhan non-fungsionalitas yang didapatkan, terdapat kebutuhan non-fungsional yang diperlukan dalam pekerjaan ini, yaitu kebutuhan dalam aspek *maintainability*. Karena Mahoni menerapkan beberapa tema berbeda yang mana masing-masing menerapkan beberapa fitur, sistem yang dibuat akan menjadi sulit dikelola apabila pengembangan hanya dilakukan dalam satu servis. Oleh karena itu, salah satu pendekatan

yang dilakukan untuk memenuhi kebutuhan *Maintainability* ini adalah dengan menerapkan arsitektur *microservice* dengan mengembangkan sistem menjadi servis-servis terpisah. Penerapan arsitektur ini juga dapat mendukung aspek *scalability* apabila dibutuhkan ke depannya.

4.2.7 Elisitasi Batasan Kebutuhan

Batasan kebutuhan merupakan aturan yang membatasi proses dari jalannya kebutuhan yang nantinya dibangun. Bentuk dari batasan kebutuhan dapat berupa berbagai macam hal seperti waktu, biaya, teknologi, peraturan, dan juga kebutuhan pengguna. Batasan kebutuhan pada pekerjaan ini tidak hanya didapatkan dari hasil elisitasi kebutuhan fungsional dan non-fungsional, tetapi juga didapatkan dengan melakukan identifikasi batasan yang dimiliki oleh pengembang ataupun pekerjaan yang dilakukan.

Pada pekerjaan ini, terdapat beberapa batasan pengembangan yang tertera pada subbab 1.4, yaitu terkait dengan *scope* pengembangan *backend* servis Mahoni dan penggunaan data *dummy*. Selain itu, terdapat batasan waktu pekerjaan dan proses bisnis sehingga tidak semua fitur yang diidentifikasi dari proses elisitasi kebutuhan sebelumnya dapat diimplementasi. Karena adanya batasan ini, perlu diketahui prioritas pengembangan fitur untuk menentukan kepentingan fitur. Pemrioritasan fitur tersebut dilakukan pada tahapan selanjutnya, yaitu pengelompokan kebutuhan.

4.2.8 Pengelompokan Kebutuhan

Setelah kebutuhan fungsional terelisitasi, kebutuhan tersebut dikelompokkan berdasarkan prioritas atau kepentingan implementasinya untuk mengetahui fitur apa yang harus diimplementasi terlebih dahulu dalam proses pengembangan. Metode yang digunakan untuk melakukan pengelompokan fitur adalah Kano *Method*. Metode ini dipilih karena selain cukup banyak digunakan untuk pengelompokan fitur berdasarkan kepentingan dan kepuasan pengguna, metode ini cukup sederhana dan dapat mengelompokkan kebutuhan menjadi beberapa kategori. Proses untuk pengumpulan data untuk pengelompokan fitur ini dilakukan dengan melakukan survei yang menggunakan kuesioner.

4.2.8.1 Desain dan Responden

Terdapat sejumlah 67 responden kuesioner, dengan 19 responden pernah menggunakan aplikasi pemantau kualitas udara, 65 responden pernah menggunakan aplikasi pendamping transportasi, dan 38 responden pernah menggunakan aplikasi yang menerapkan sistem poin dan kupon. Responden merupakan warga negara Indonesia berdomisili di Jabodetabek yang berusia antara 18–45 tahun. Kuesioner dibuat menggunakan platform Google Form dan responden mengisi kuesionernya secara *online*. Sebelum mengisi kuesioner, responden diminta untuk mencoba menggunakan purwarupa fidelitas tinggi aplikasi Mahoni yang telah disediakan. Kemudian, responden tersebut mengisi pendapat mereka mengenai kandidat fitur yang terdapat pada purwarupa.

Pertanyaan yang terdapat pada kuesioner merupakan pertanyaan Kano dengan menggunakan skala ordinal serta terdapat pertanyaan tambahan berbentuk *open-ended* sebagai masukan dalam pengembangan. Setiap pertanyaan dalam kuesioner Kano mewakilkan satu kandidat fitur. Terdapat 12 kandidat fitur yang ditanyakan, yaitu riwayat kualitas udara (F1), informasi dan pencarian kualitas udara (F2), prediksi kualitas udara (F3), rekomendasi kesehatan (F4), artikel (F5), *push-notification* (F6), informasi perjalanan transportasi (F7), riwayat perjalanan (F8), jumlah poin (F9), riwayat penukaran poin (F10), daftar dan detail kupon tersedia (F11), dan daftar dan detail kupon dimiliki (F12). Salah satu pertanyaan fitur dengan bentuk pertanyaan Kano yang ditanyakan dapat dilihat pada Tabel 4.5 dan lebih lengkapnya pada Lampiran 3.

Tabel 4.5 Pertanyaan Kuesioner Kano

Jenis Pertanyaan	Pertanyaan	Jawaban
Fungsional	Jika ada fitur rekomendasi yang memberikan rekomendasi kegiatan berdasarkan kualitas udara saat ini, bagaimana perasaan Anda?	1. Suka 2. Mengharapkan 3. Netral 4. Dapat menoleransi 5. Tidak suka
Disfungsional	Jika tidak ada fitur rekomendasi yang memberikan rekomendasi kegiatan berdasarkan kualitas udara saat ini, bagaimana perasaan Anda?	1. Suka 2. Mengharapkan 3. Netral 4. Dapat menoleransi 5. Tidak suka

Pertanyaan kuesioner Kano yang ditanyakan beserta jawabannya diterjemahkan dari bahasa Inggris di mana padanan opsi jawaban yang digunakan adalah suka (*like*), mengharapkan (*must-be*), netral (*neutral*), dapat menoleransi (*live with*), dan tidak suka (*dislike*). Terdapat dua jenis pertanyaan untuk satu fitur yaitu pertanyaan fungsional dan pertanyaan disfungsional. Pertanyaan yang disampaikan pada jenis pertanyaan fungsional menanyakan sisi positif atau aspek yang ingin dicapai dari fitur yang ditanyakan, sedangkan pertanyaan disfungsional menanyakan kebalikan sifat dari pertanyaan fungsional. Untuk setiap kandidat fitur yang dipilih, ditanyakan kedua jenis pertanyaan tersebut pada kuesioner.

4.2.8.2 Analisis Data dan Hasil

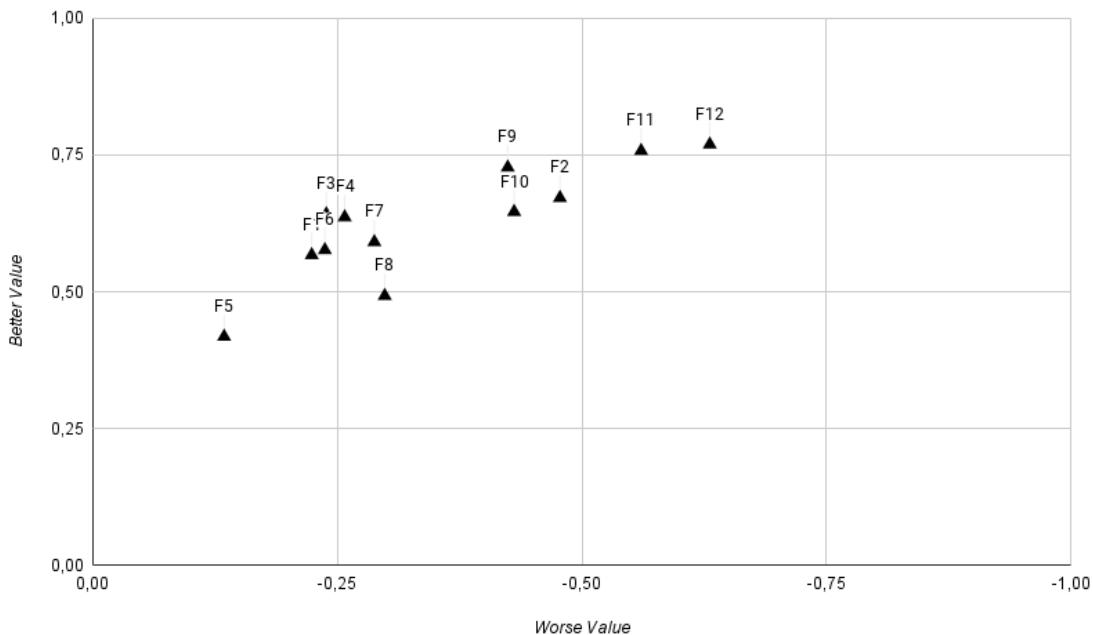
Setiap jawaban responden untuk setiap pertanyaan dicatat, dikelompokkan, dan dihitung jumlah jawaban yang termasuk ke dalam masing-masing kategori yang ada. Pengelompokan dan perhitungan dilakukan dengan menggunakan acuan tabel seperti pada Tabel 2.3. Hasil dari perhitungan evaluasi kuesioner Kano tertera pada Lampiran 15 dan dapat dilihat rangkumannya pada Tabel 4.6.

Tabel 4.6 Perhitungan Kuesioner Kano

Fitur	M	O	A	I	R	Q	Cat	+	-
F1	1	14	24	28	0	0	I	0,57	-0,22
F2	5	27	18	17	0	0	P	0,67	-0,48
F3	1	15	28	23	0	0	A	0,64	-0,24
F4	1	16	26	23	0	1	A	0,64	-0,26
F5	1	8	20	38	0	0	I	0,42	-0,13
F6	0	14	20	25	4	4	I	0,58	-0,24
F7	1	18	21	26	0	1	I	0,59	-0,29
F8	2	18	15	32	0	0	I	0,49	-0,30
F9	2	26	22	16	1	0	P	0,73	-0,42
F10	4	24	18	19	1	1	P	0,65	-0,43
F11	2	35	15	14	0	1	P	0,76	-0,56
F12	4	37	13	11	1	1	P	0,77	-0,63

Pada tabel perhitungan kategori Kano terlihat persebaran jumlah jawaban untuk setiap kategorinya. Dari sini, kategori untuk setiap fitur didapatkan dari modus jumlah jawaban yang dimiliki kategori (Cat) tersebut apakah *must-be* (M), *one-dimensional* (O), *attractive*

(A), atau *indifferent* (I). Namun, pengelompokan lebih lanjut dapat dilakukan dengan melakukan perhitungan skor positif (+) dan skor negatif (-) menggunakan formula yang tertera pada Persamaan 2.2 dan Persamaan 2.3. Nilai skor positif (+) dan skor negatif (-) tersebut kemudian dipetakan ke dalam Kano Diagram yang dapat dilihat pada Gambar 4.5.



Gambar 4.6 Pemetaan Hasil Kuesioner pada Kano Diagram

Dilihat dari pemetaan Kano Diagram, terdapat: dua fitur yaitu F11 dan F12 yang tergolong kategori *one-dimensional* (kuadran *worse-value* -0,50–(-1), *better-value* 0,5–1); delapan fitur yaitu F1, F2, F3, F4, F6, F7, F9, dan F10 tergolong kategori *attractive* (kuadran *worse-value* 0–(-0,5), *better-value* 0,5–1); dua fitur yaitu F5 dan F8 tergolong kategori *indifferent* (kuadran *worse-value* 0–(-0,5), *better-value* 0–0,5); dan tidak ada fitur yang tergolong kategori *must-be* (kuadran *worse-value* -0,50–(-1), *better-value* 0–0,5). Hal tersebut dapat disebabkan oleh kesalahpahaman pengguna mengenai jawaban akibat kemungkinan translasi yang kurang sesuai sehingga responden memiliki kecenderungan dalam memilih jawaban tertentu. Hingga pekerjaan ini selesai dilakukan, belum adanya penelitian mengenai translasi Kano *Method* ke dalam bahasa Indonesia yang menyebabkan kemungkinan ketidakakuratan terjemahan pada pilihan jawaban Kano yang digunakan.

Berdasarkan hasil pemetaan yang dilakukan, diputuskan sembilan fitur yang dipilih untuk diimplementasikan berupa servis. Semua fitur yang berada pada kategori *one-dimensional* dipilih karena merupakan fitur yang paling penting dan memenuhi kepuasan pengguna. Tidak semua fitur yang berada pada kategori *attractive* dipilih, yaitu tidak dipilihnya fitur prediksi kualitas udara (F3) dan *push-notification* (F6). Fitur prediksi kualitas udara tidak dipilih karena keterbatasan data yang ada dan menjadi fokus pekerjaan tersendiri, serta fitur *push-notification* tidak dipilih karena termasuk pada ranah *mobile-development* yang tidak termasuk ke dalam lingkup pekerjaan. Kemudian, fitur artikel (F5) yang berada pada kategori *indifferent* tidak dipilih karena tidak membawa nilai lebih untuk kepuasan pengguna, sedangkan fitur riwayat perjalanan (F8) dipilih untuk diimplementasikan karena menunjang proses bisnis.

Kesembilan fitur yang dipilih yaitu F1, F2, F4, F7, F8, F9, F10, F11, dan F12 selanjutnya menjadi penyusun implementasi servis-servis Mahoni. Tidak hanya fitur hasil *requirement gathering* yang diimplementasikan, tetapi terdapat beberapa fitur tambahan yang juga diterapkan dalam servis untuk pengelolaan lebih lanjut. Sebelum melakukan implementasi, fitur-fitur tersebut diterjemahkan kembali menjadi kebutuhan pengguna dan dikembangkan lebih mendalam secara teknis. Pembahasan mengenai hal tersebut terdapat pada subbab selanjutnya yaitu perancangan servis.

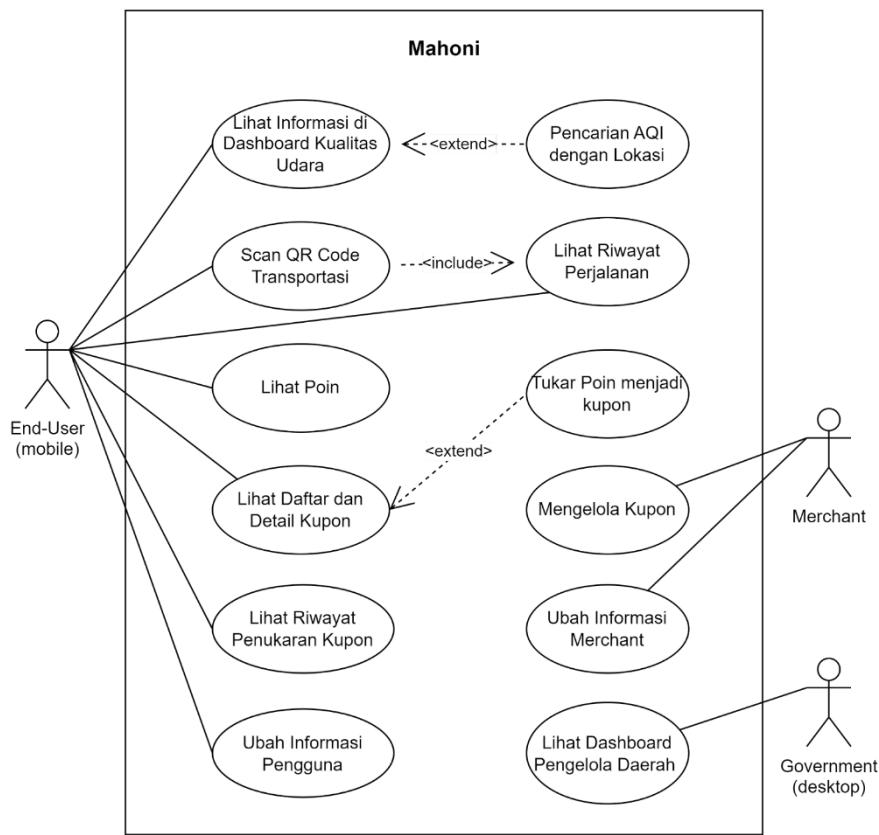
4.3 Perancangan Servis

Untuk dapat mengimplementasikan fitur yang telah ditentukan, perlu adanya servis yang melayani fitur tersebut. Namun seperti yang tertera pada batasan penelitian, pekerjaan ini tidak berusaha untuk mengimplementasikan fitur pada pengembangan *mobile* alih-alih pengembangan servisnya. Perancangan servis dimulai dengan memetakan fungsionalitas fitur terhadap pengguna yang akan menggunakan sistem. Pengembangan servis Mahoni dilakukan atas penerapan arsitektur *microservice* sehingga diperlukan untuk memetakan fitur berdasarkan domainnya dalam servis terpisah.

4.3.1 Pemetaan Kebutuhan dan Sistem

Fitur yang dipilih pada tahap praperancangan dikembangkan menjadi *use case* yang visualisasinya tersedia pada Gambar 4.7 yaitu *use case* diagram Mahoni. Terdapat tiga peran dalam diagram tersebut yaitu pengguna umum, mitra usaha, dan pengelola daerah

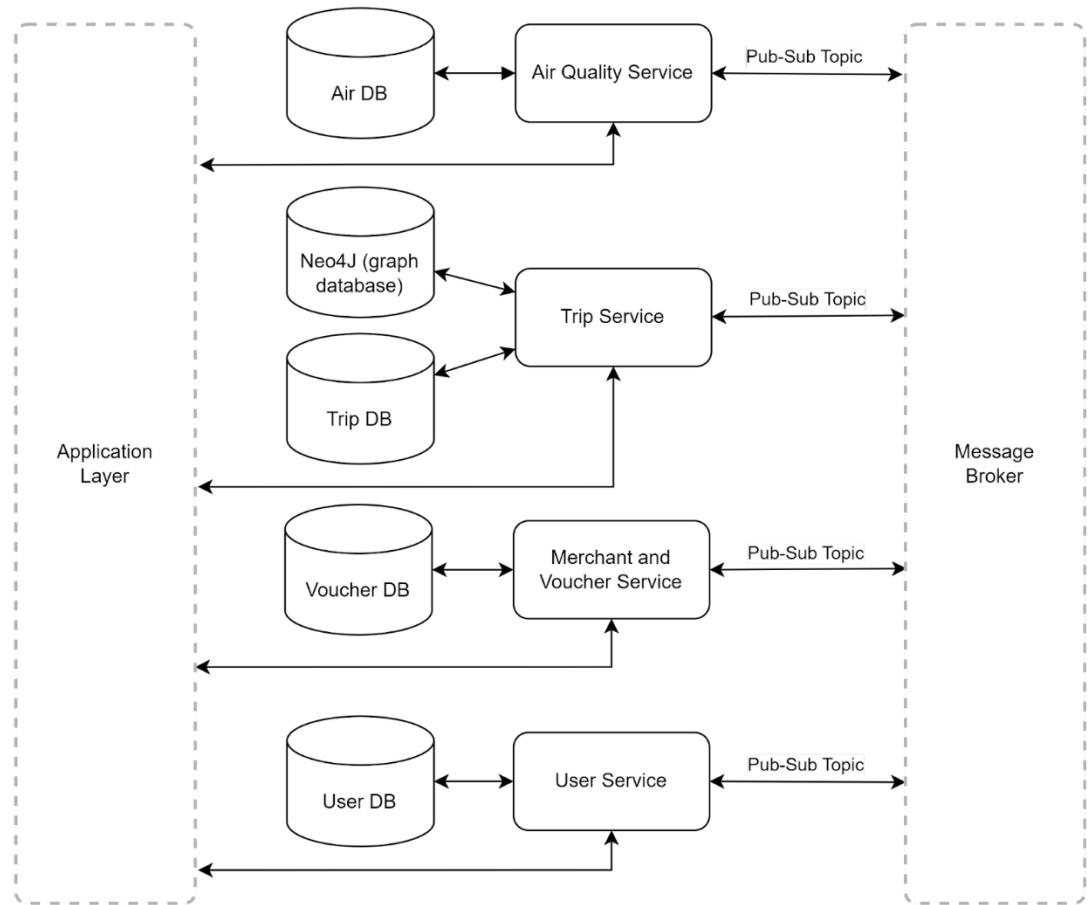
yang dapat berinteraksi dengan sepuluh *use case*. Pengguna umum (*end-user*) dapat menjalankan *use case* lihat informasi di *dashboard* kualitas udara (UC1), pencarian AQI dengan lokasi (UC2), *scan qr code* transportasi (UC3), lihat riwayat perjalanan (UC4), lihat poin (UC5), lihat daftar dan detail kupon (UC6), tukar poin menjadi kupon (UC7), lihat riwayat penukaran kupon (UC8), dan ubah informasi pengguna (UC9). Mitra usaha (*merchant*) dapat menjalankan *use case* mengelola kupon (UC10) dan ubah informasi *merchant* (UC11). Pengelola daerah (*government*) dapat menjalankan *use case* lihat *dashboard* pengelola daerah (UC12).



Gambar 4.7 Use Case Diagram Mahoni

Tidak semua fitur dipetakan secara langsung ke dalam *use case* yaitu: fitur F1 dan F4 secara bersama-sama masuk ke dalam *use case* UC1, fitur F11 dan F12 tergabung dalam *use case* UC6, fitur F7 disediakan dalam *use case* UC3, dan UC7 tidak berasal dari fitur hasil pengelompokan kebutuhan tetapi merupakan kebutuhan dasar pengguna umum. Khusus untuk *use case* untuk pengelola daerah yaitu UC10 tidak diimplementasikan dalam pekerjaan ini tetapi pada implementasi arsitektur *big data* pada bab 6.

Setelah dilakukan pemetaan *use case*, dilakukan perancangan arsitektur *microservice* sistem Mahoni yang dibangun atas servis-servis terpisah seperti yang dapat dilihat pada Gambar 4.8. Setiap servis bertanggung jawab dalam satu domain tema dari Mahoni yaitu kualitas udara, perjalanan transportasi umum, dan poin dan kupon. Pengembangan *use case* yang tertera pada *use case* diagram diimplementasikan ke dalam beberapa servis tersebut.



Gambar 4.8 Arsitektur *Microservice* Mahoni

Servis kualitas udara bertanggung jawab dalam penyediaan dan pengelolaan informasi berkaitan dengan kualitas udara dan mengimplementasikan *use case* U1 dan U2. Servis perjalanan transportasi umum bertanggung jawab dalam pengelolaan pencatatan penggunaan transportasi umum pengguna dan mengimplementasikan *use case* U3 dan U4. Servis kupon dan mitra usaha bertanggung jawab dalam pengelolaan kupon dan pengguna mitra usaha serta mengimplementasikan *use case* U5, U6, U7, U8, dan U9.

Servis pengguna tidak menerapkan *use case* yang diidentifikasi melainkan bertanggung jawab untuk mengelola data pengguna umum itu sendiri.

4.3.2 Pemilihan Teknologi

Sebelum mulai merancang servis, ditentukan teknologi yang dipakai dalam pengembangan karena memengaruhi proses perancangan. Setiap servis terhubung dengan *database* untuk penyimpanan data dan juga ke *message broker* sebagai jalur komunikasi antar servis. Pemilihan dan pembahasan teknologi *message broker* tidak dijelaskan dalam pekerjaan ini tetapi pada bab 5 mengenai implementasi arsitektur *event-driven*. Oleh karena itu pemilihan teknologi dilakukan kepada bahasa pemrograman dan pemilihan *framework web* untuk implementasi servis dan *database* yang digunakan.

Bahasa pemrograman yang digunakan adalah Java karena menyediakan berbagai banyak pilihan *library* untuk menunjang pengembangan servis juga untuk pengembangan arsitektur *event-driven* dan *big-data* pada bab 5 dan bab 6. Kemudian, pemilihan *framework web* didasarkan pada ekosistem *framework* dan apakah penulis telah menguasai *framework* yang dipilih. Dengan demikian, *framework web* dengan bahasa pemrograman Java yang digunakan untuk membangun seluruh servis pada sistem Mahoni adalah Spring Boot.

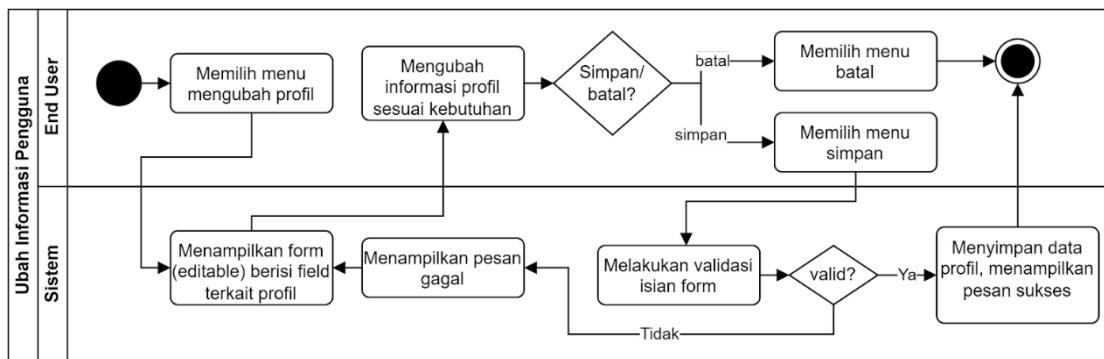
Spring Boot merupakan *framework web* yang cukup terkenal dan sudah banyak digunakan pada oleh perusahaan-perusahaan sebagai standar *framework web* yang memberikan performa baik dan keamanan tinggi. Selain itu, penulis sudah pernah mempelajari *framework* ini di mata kuliah sebelumnya sehingga tidak perlu menghabiskan waktu untuk mempelajari ulang bagaimana *framework* ini bekerja.

Setelah penentuan teknologi untuk servis, selanjutnya adalah *database* yang digunakan. Terdapat dua jenis *database* yang dibutuhkan pada pekerjaan, yaitu *relational database* untuk menyimpan data terstruktur seluruh servis dan *graph database* untuk pemetaan lokasi dalam servis perjalanan. PostgreSQL dipilih sebagai *relational database* karena merupakan pilihan populer untuk berbagai jenis aplikasi sebab teknologinya yang matang dan dukungan komunitas yang aktif (*PostgreSQL: About*, n.d.). Neo4j dipilih sebagai *graph database* karena Neo4j menawarkan teknologi yang matang dan banyak

digunakan, menyediakan beragam fitur, serta memiliki ekosistem dan dukungan komunitas yang kuat (Neo4j, 2021).

4.3.3 Perancangan Servis Pengguna

Servis pengguna (*user service*) merupakan servis yang menangani permintaan terkait pengguna, dalam hal ini adalah pengguna umum atau *end-user*. Terdapat satu *use case* yang ditangani oleh servis ini yaitu ubah informasi pengguna (UC9). Seperti alur yang tertera pada Gambar 4.9, pengguna mengubah informasinya dengan memilih menu ubah profil, kemudian mengubah informasi sesuai kebutuhan, lalu menyimpannya. Aktivitas tersebut dilakukan *end-user* melalui interaksi kepada *user interface* aplikasi Mahoni dan direspon oleh sistem aplikasi *mobile* maupun servis.



Gambar 4.9 Activity Diagram Mengubah Profil

Aktivitas *use case* mengubah informasi pengguna kemudian diterjemahkan menjadi *table* untuk diimplementasikan dengan mengidentifikasi entitas dan aksi yang dikerjakan. Terdapat satu entitas yang berperan dalam aktivitas tersebut yaitu pengguna umum. Aksi yang kerjakan adalah mengubah entitas tersebut. Aksi mengubah entitas tidak memerlukan informasi apa pun kecuali entitas yang diubahnya. Oleh karena itu, untuk domain ini hanya dibuat satu *table* yang berperan dalam aktivitas tersebut, yaitu *table users* pengguna umum aplikasi.



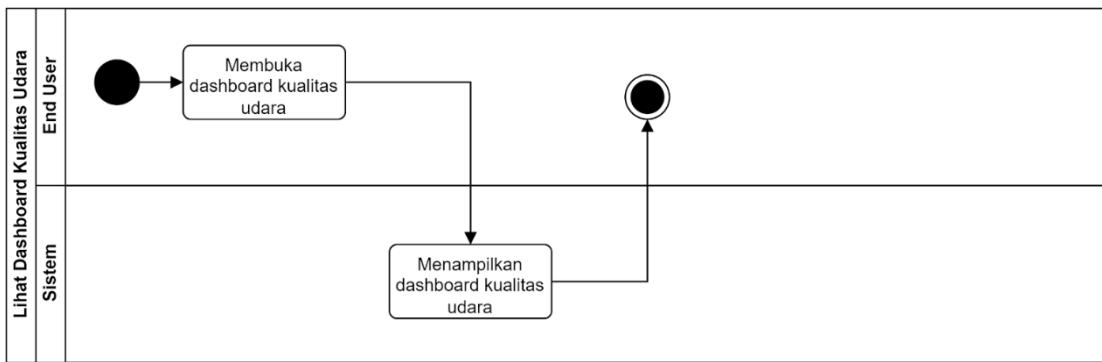
Gambar 4.10 ER Diagram Servis Pengguna

Informasi seputar pengguna yang dibutuhkan dalam menyelesaikan *use case* merupakan kolom dari *table users* yaitu: *identifier id*; data diri seperti **username**, **name**, **email**, **sex**, dan **year_of_birth**; dan **point** untuk pencatatan poin. Kolom data diri yang tertera digunakan sebagai tanda pengenal pengguna dan dibutuhkan untuk analisis lanjut dalam *dashboard use case* U10. Keterangan mengenai tipe data yang dipakai untuk setiap kolom dapat dilihat dalam Gambar 4.10.

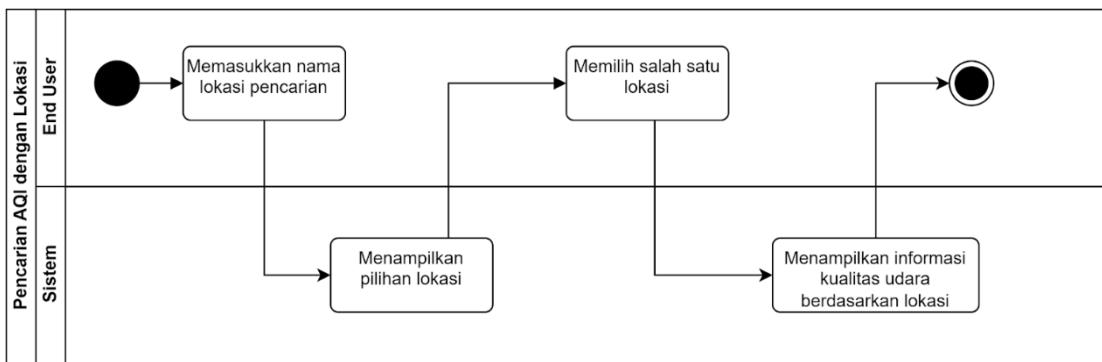
4.3.4 Perancangan Servis Kualitas Udara

Servis kualitas udara (*air quality service*) merupakan servis yang menangani permintaan terkait informasi pada *dashboard* kualitas udara di aplikasi seperti informasi kualitas udara, riwayat kualitas udara, dan rekomendasi kesehatan. Terdapat dua *use case* yang ditangani oleh servis ini yaitu lihat informasi di *dashboard* kualitas udara (UC1) dan pencarian AQI dengan kota (UC2).

Seperti alur yang tertera pada Gambar 4.11, pengguna melihat informasi pada *dashboard* dengan memilih menu *dashboard* kualitas udara pada aplikasi. Informasi kualitas udara yang ditampilkan berdasarkan lokasi pengguna saat itu menggunakan GPS ataupun melalui pencarian lokasi. Gambar 4.12 menunjukkan alur pengguna mencari AQI berdasarkan lokasi dengan memasukkan nama lokasi kemudian memilih lokasi yang tersedia untuk ditampilkan informasinya. Aktivitas tersebut dilakukan *end-user* melalui interaksi kepada *user interface* aplikasi Mahoni dan direspon oleh sistem aplikasi *mobile* maupun servis.

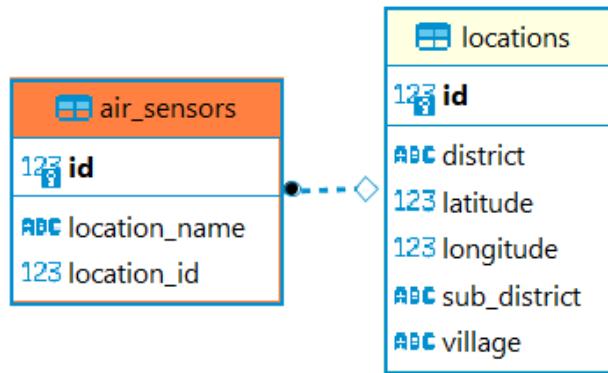


Gambar 4.11 Activity Diagram Lihat Informasi Kualitas Udara



Gambar 4.12 Activity Diagram Pencarian AQI dengan Lokasi

Aktivitas *use case* UC1 dan UC2 kemudian diterjemahkan menjadi *table* untuk diimplementasikan dengan mengidentifikasi entitas dan aksi yang dikerjakan. Entitas yang berperan dalam aktivitas tersebut yaitu kualitas udara dan lokasi. Kualitas udara didapatkan dari berbagai sensor udara yang tersebar di penjuru lokasi. Aksi yang kerjakan adalah melihat dan mencari informasi terkait entitas kualitas udara sehingga tidak memerlukan informasi tambahan apa pun dari entitas tersebut. Entitas kualitas udara tidak diimplementasikan langsung menjadi sebuah *table* karena sifat datanya yang temporal dan sementara sehingga diterapkan dalam bentuk lain yaitu KTable yang dijelaskan lebih lanjut pada subbab 5.3.2.2. Oleh karena itu, untuk domain ini *table* yang dibuat adalah sensor udara **air_sensors** dan lokasi **locations**.



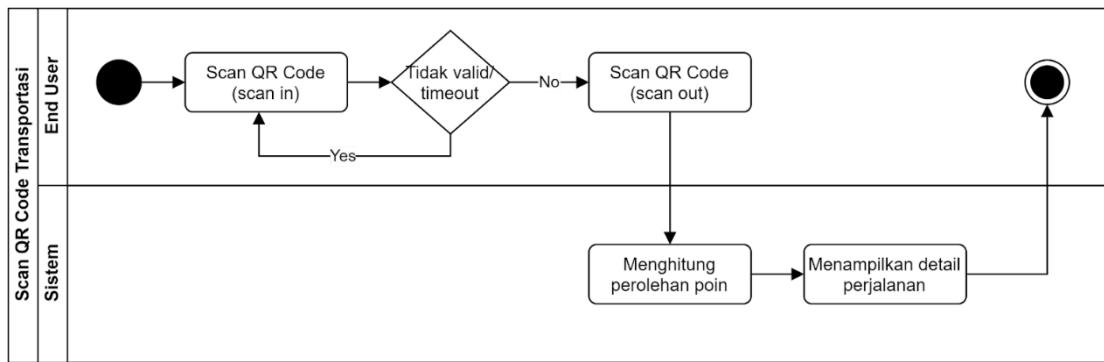
Gambar 4.13 ER Diagram Servis Kualitas Udara

Kolom dari *table locations* yaitu: *identifier id*; lokasi seperti **district** (kota atau kabupaten), **sub_district** (kecamatan), dan **village** (desa atau kelurahan); dan geolokasi untuk pencarian lokasi terdekat dengan **longitude** dan **latitude**. Di sisi lain, kolom dari *table air_sensors* adalah: *identifier id*, nama lokasi **location_name**, dan **location_id**. Keterangan mengenai tipe data yang dipakai untuk setiap kolom dapat dilihat dalam Gambar 4.13.

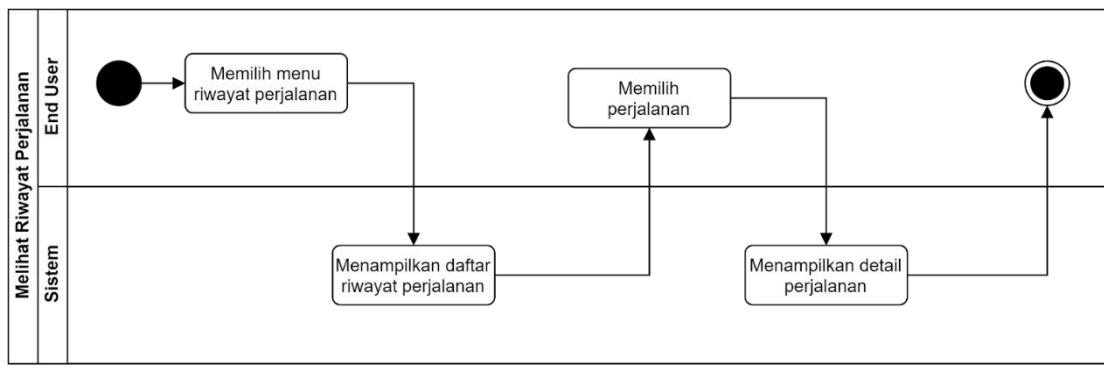
4.3.5 Perancangan Servis Perjalanan

Servis perjalanan (*trip service*) merupakan servis yang menangani pencatatan penggunaan transportasi umum oleh pengguna dan perolehan poin. *Use case* yang diimplementasikan oleh servis ini adalah *scan qr code* transportasi (UC 3) dan lihat riwayat perjalanan (UC 4).

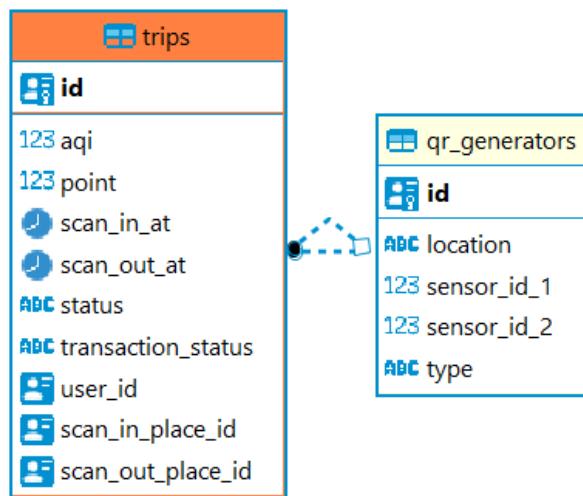
Seperti alur yang tertera pada Gambar 4.14, pengguna melakukan pencatatan penggunaan transportasi umum dengan melakukan *scan in* dan *scan out qr code*. Gambar 4.15 memperlihatkan alur melihat riwayat perjalanan yaitu dengan memilih menu riwayat perjalanan kemudian memilih perjalanan yang ingin diketahui informasinya. Aktivitas tersebut dilakukan *end-user* melalui interaksi kepada *user interface* aplikasi Mahoni dan *direspons* oleh sistem aplikasi *mobile* maupun servis.



Gambar 4.14 Activity Diagram Scan QR Code Transportasi



Gambar 4.15 Activity Diagram Melihat Riwayat Perjalanan



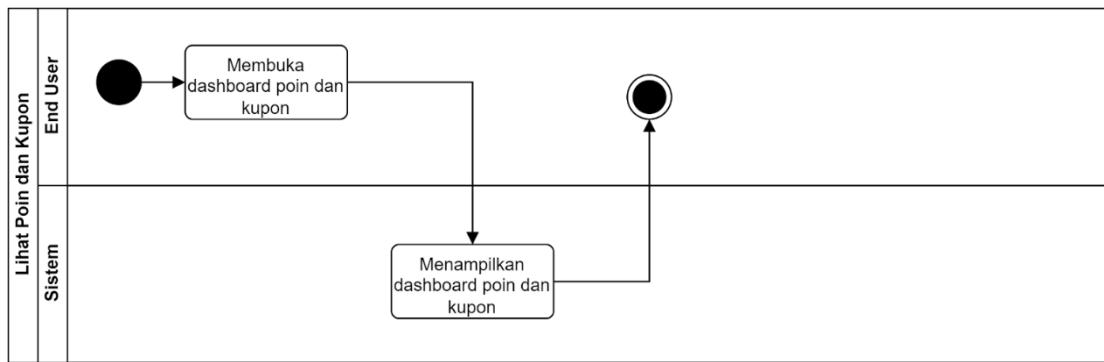
Gambar 4.16 ER Diagram Servis Perjalanan

Aktivitas *use case* UC3 dan UC4 kemudian diterjemahkan menjadi *table* untuk diimplementasikan dengan mengidentifikasi entitas dan aksi yang dikerjakan. Entitas yang berperan dalam aktivitas tersebut yaitu generator untuk *qr code* dan aksi yang kerjakan adalah perjalanan transportasi umum. *Qr code* sendiri tidak disimpan karena bersifat temporal dan sementara. Oleh karena itu, untuk domain ini *table* yang dibuat adalah generator **qr_generators** dan perjalanan **trips**.

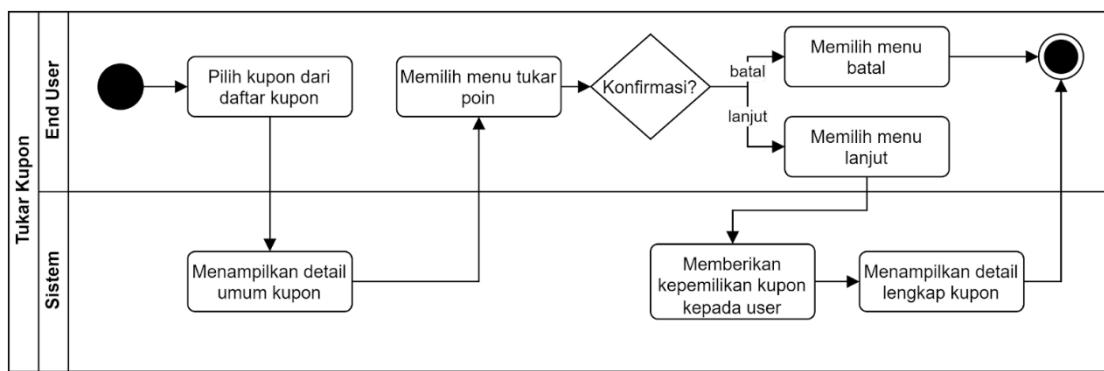
Kolom dari *table trips* yaitu: *identifier id*; *id* pengguna **user_id** yang merujuk pada **id** di *table users*; informasi perjalanan seperti tempat *scan scan_in_place_id* dan *scan_out_place_id* yang merupakan *foreign key* ke *table qr_generators*, waktu *scan scan_in_at* dan *scan_out_at*, serta status perjalanan **status**; dan informasi perolehan poin yaitu status transaksi **transaction_status**, **aqi**, dan **point**. Kolom dari *table qr_generators* adalah: *identifier id*, nama lokasi **location**, tipe transportasi umum **type**, dan *id* sensor udara **sensor_id_1** dan **sensor_id_2** yang merujuk pada **id** di *table air_sensors*. *Id* sensor udara disimpan dalam kolom tersebut karena dalam perhitungan jumlah poin besarnya nilai AQI dari tempat perjalanan menjadi pertimbangan. Keterangan mengenai tipe data yang dipakai untuk setiap kolom dapat dilihat dalam Gambar 4.16.

4.3.6 Perancangan Servis Kupon dan Mitra Usaha

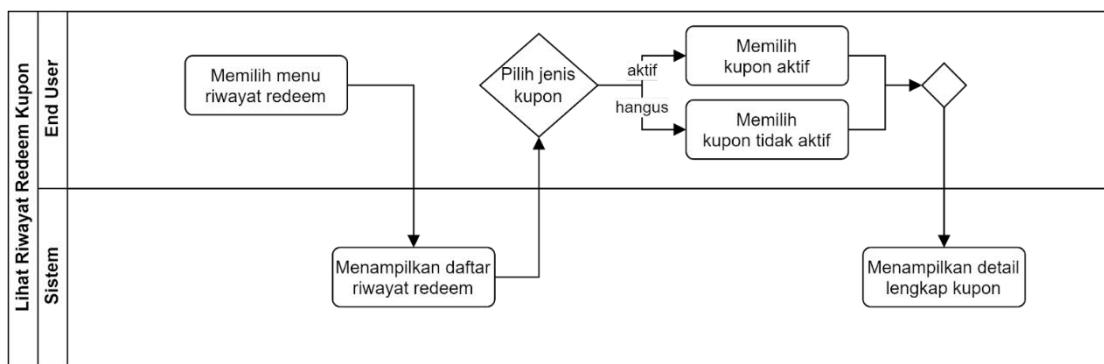
Servis kupon dan mitra usaha (*voucher and merchant service*) merupakan servis yang menangani permintaan terkait penukaran poin menjadi kupon oleh pengguna umum serta pengelolaan kupon oleh mitra usaha. Aktivitas *use case* UC5 dan UC6 tertera pada Gambar 4.17, yaitu pengguna membuka halaman utama poin dan kupon kemudian melihat informasi terkait, apakah poin ataupun kupon. Aktivitas *use case* UC7 tertera pada Gambar 4.18, yaitu memilih kupon yang ingin dibeli dari daftar kupon kemudian memilih menu tukar poin dan lanjut. Aktivitas *use case* UC8 tertera pada Gambar 4.19, yaitu memilih menu riwayat *redeem* (pembelian kupon) kemudian memilih kupon berdasarkan statusnya untuk melihat informasi kuponnya.



Gambar 4.17 Activity Diagram Lihat Poin dan Kupon



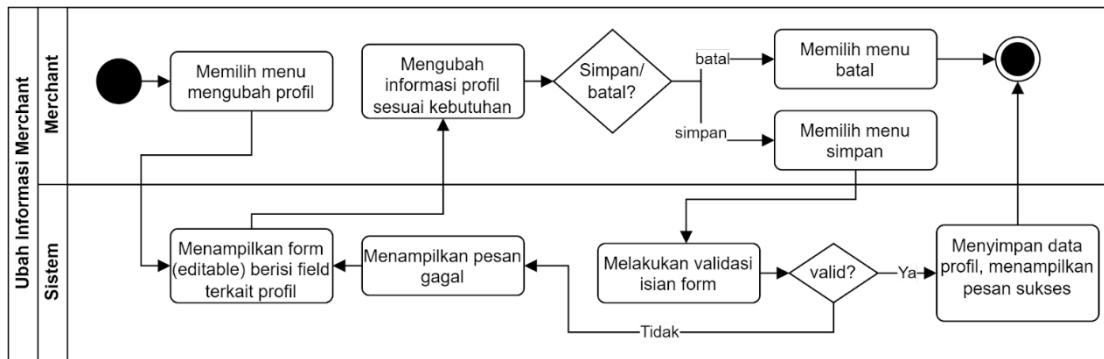
Gambar 4.18 Activity Diagram Tukar Poin



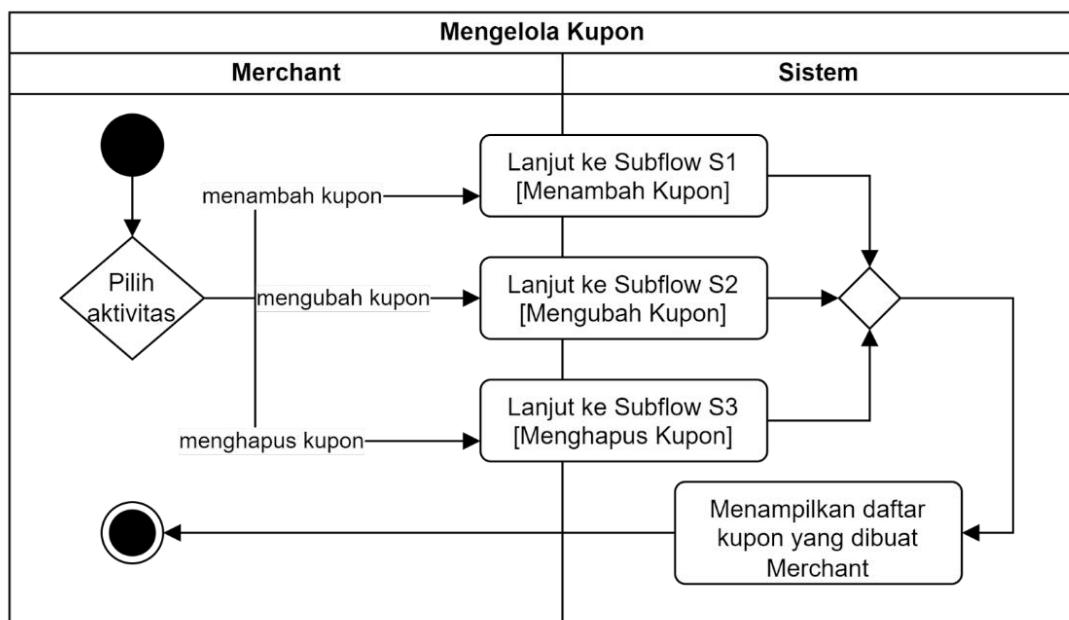
Gambar 4.19 Activity Diagram Lihat Riwayat Redeem Kupon

Selain pengguna umum, segala aktivitas yang dilakukan mitra usaha dilayani oleh servis ini. Terdapat dua *use case* yang ditangani oleh servis ini yaitu kelola kupon (UC10) dan ubah informasi *merchant* (UC11). Seperti alur yang tertera pada Gambar 4.20, mitra usaha mengubah informasinya dengan memilih menu ubah profil, kemudian mengubah

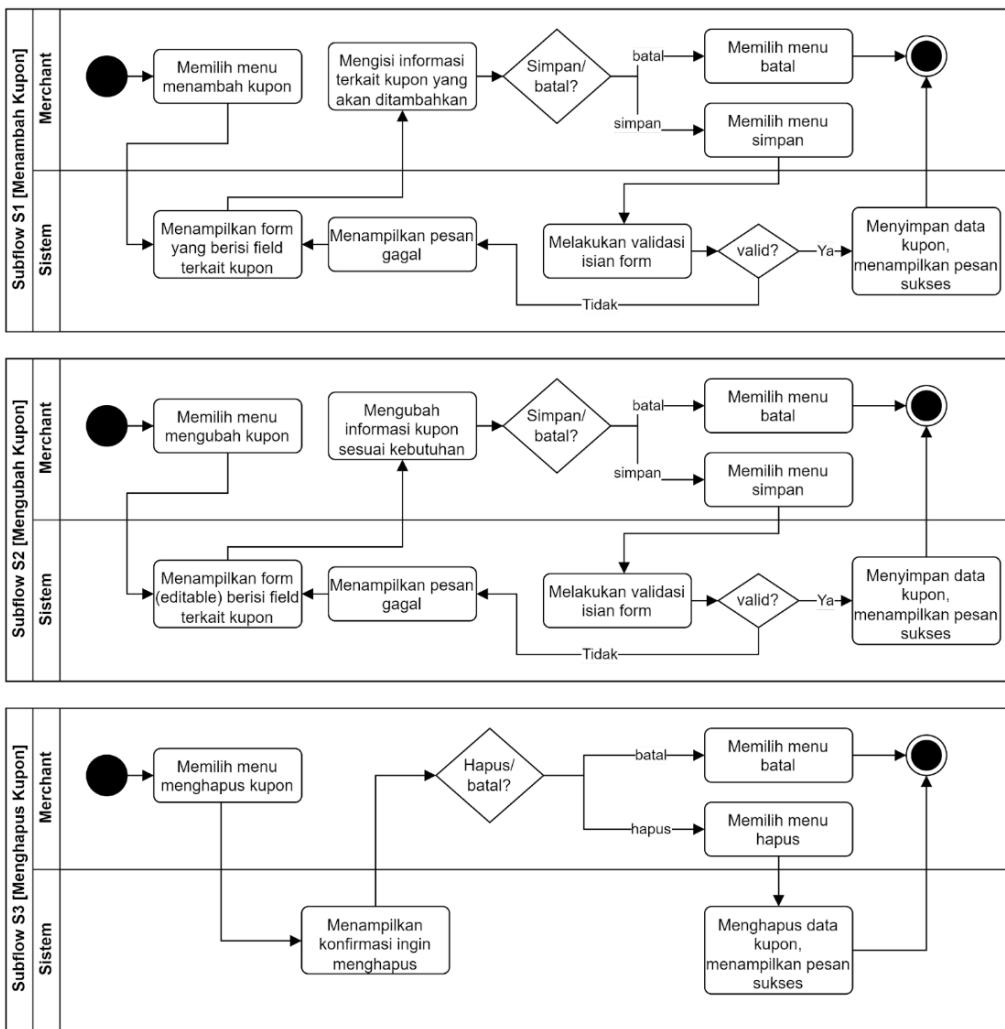
informasi sesuai kebutuhan, lalu menyimpannya. Gambar 4.21 menunjukkan alur mengelola kupon yang terdiri dari tiga *subflow*: Subflow S1 menambah kupon, Subflow S2 mengubah kupon, Subflow S3 menghapus kupon. Alur masing-masing kupon terdapat dalam Gambar 4.22



Gambar 4.20 Activity Diagram Ubah Informasi Merchant

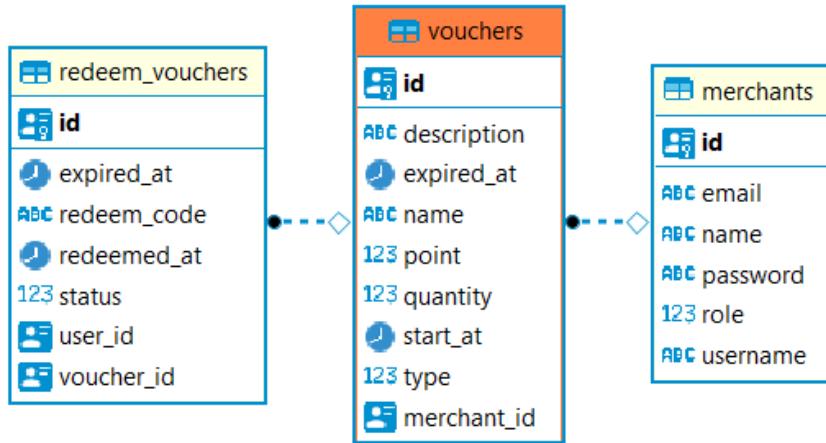


Gambar 4.21 Activity Diagram Mengelola Kupon



Gambar 4.22 Activity Diagram Subflow S1-S3 Mengelola Kupon

Aktivitas *use case* UC5, UC6, UC7, UC8, UC10, dan UC11 kemudian diterjemahkan menjadi *table* untuk implementasinya dengan mengidentifikasi entitas dan aksi yang dikerjakan. Terdapat tiga entitas yang berperan dalam aktivitas tersebut yaitu pengguna umum, mitra usaha, dan kupon. Entitas pengguna umum diimplementasikan di servis pengguna sehingga tidak termasuk pada servis ini. Aksi yang kerjakan adalah melihat informasi poin, kupon, dan riwayat *redeem*; tukar kupon; dan mengelola kupon. Aksi tukar kupon membutuhkan informasi tambahan dari pengguna umum maupun kupon seperti tanggal penukaran dan kode unik. Oleh karena itu, untuk domain ini *table* yang dibuat adalah mitra usaha **merchants**, kupon **vouchers**, dan tukar kupon **redeem_vouchers**.



Gambar 4.23 ER Diagram Servis Kupon dan Mitra Usaha

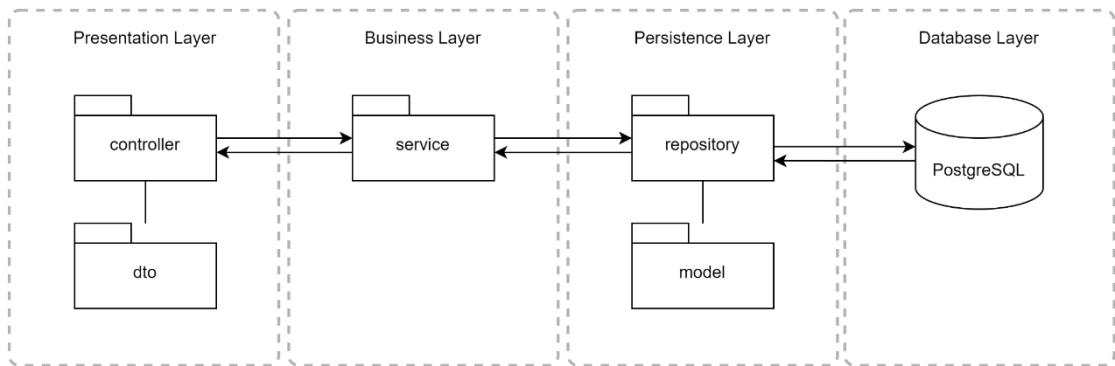
Kolom dari *table merchants* yaitu: *identifier id* dan data diri seperti **username**, **name**, **email**, **password**, dan **role**. Kolom dari *table vouchers* adalah: *identifier id*; informasi kupon seperti **name**, **description**, tipe kupon **type**, jumlah poin untuk penukaran kupon **point**, tanggal berlaku kupon **start_at**, tanggal kadaluwarsa kupon **expired_at**, dan **merchant_id** yang merujuk pada **id** di *table merchants*; dan jumlah kupon tersedia **quantity**. Kolom dari *table redeem_vouchers* yaitu: *identifier id*; *foreign key voucher_id* yang merujuk pada **id** di *table vouchers* dan **user_id** yang merujuk pada **id** di *table users*; informasi tukar kupon yaitu **status**, tanggal tukar kupon **redeemed_at**, dan tanggal kadaluwarsa kupon **expired_at**. Keterangan mengenai tipe data yang dipakai untuk setiap kolom dapat dilihat dalam Gambar 4.23.

4.4 Pengembangan Servis

Rancangan servis-servis Mahoni yang telah dibuat selanjutnya diimplementasikan menjadi servis yang dapat melayani fitur yang telah ditentukan. Pengembangan servis dilakukan dengan membuat *project* Spring Boot Mahoni yang di dalamnya terdapat masing-masing *project* untuk servis. Dokumentasi setiap servis tersebut terdapat pada Github Mahoni¹⁴. Arsitektur dari setiap servis mengikuti arsitektur *project* Spring Boot yang mana terdiri dari empat *layer*, yaitu *presentation layer*, *business layer*, *persistence*

¹⁴ <https://github.com/Mahoni-Smart-City/mahoni-core>

layer, dan *database layer*. Masing-masing hubungan antar *layer* dapat dilihat pada Gambar 4.24.



Gambar 4.24 *Layer* Setiap Servis

Presentation layer merupakan *layer* yang mengatur interaksi sistem dengan pengguna. Interaksi yang dilakukan berupa *request* HTTP yang menerima data JSON. *Package* dalam pengembangan yang termasuk dalam layer ini adalah **controller** dan **dto**. Komponen *request* HTTP baik *method* dan *endpoint* serta *function* yang melayaninya diatur di dalam *package controller*. *Request* tersebut diterjemahkan menjadi *class* yang disediakan dalam *package dto*. Layanan atau pun *function* yang mengolah *request* dari **controller** terdapat dalam *business layer*.

Business layer merupakan layer yang berisikan *logic* dan aturan dalam implementasi proses bisnis. Kumpulan layanan yang melayani permintaan dari *controller* terdapat dalam *package service*. Umumnya, satu permintaan ditanggapi oleh satu *function* yang mengerjakan *task* yang spesifik. Implementasi dari *function* tersebut sering membutuhkan manipulasi dan pengolahan data yang disediakan oleh layanan dalam *persistence layer*.

Persistence layer memberikan abstraksi untuk mengakses maupun memanipulasi data dalam *database layer*. Layanan abstraksi tersebut disediakan oleh *interface* yang berada dalam *package repository* dan *table* dalam *database* direpresentasikan oleh *object* yang tersedia dalam *package model*.

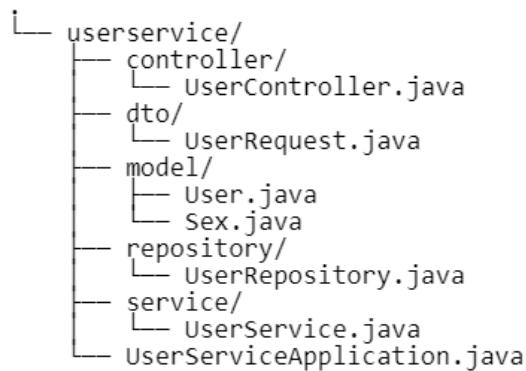
Database layer merujuk kepada lingkungan *database* asli yang digunakan. Seperti yang telah disebutkan di bagian pemilihan teknologi, *database* yang digunakan untuk

pengembangan ini adalah PostgreSQL. Akses maupun manipulasi data yang terdapat dalam *database* dikelola oleh *persistence layer*.

Setiap pengembangan servis memiliki *package controller, dto, service, repository, dan model* sebagai implementasi *layers* yang telah disebutkan sebelumnya. Masing-masing *package* memiliki *class* yang bertanggung jawab terhadap fungsi *layer* dan mewujudkan rancangan *activity diagram* maupun *ER diagram*. Penjelasan pengembangan servis-servis terdapat di subbab selanjutnya.

4.4.1 Pengembangan Servis Pengguna

Ilustrasi struktur *layer project* untuk servis pengguna dapat dilihat pada Gambar 4.25. Dalam servis ini hanya terdapat satu *package* servis yaitu **userservice**.



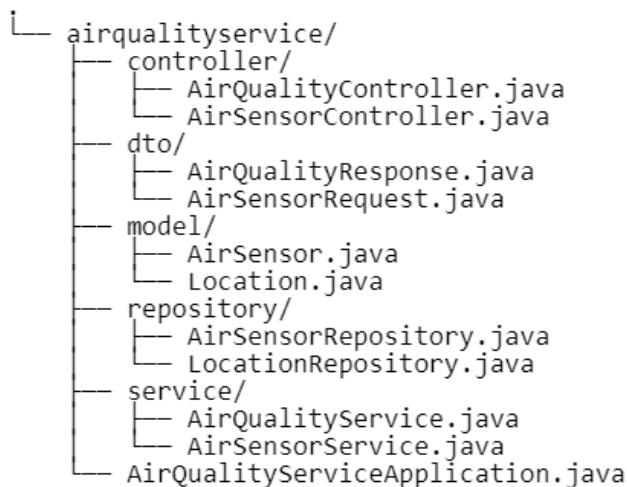
Gambar 4.25 Struktur Pohon Servis Pengguna

Penerapan peran sistem dalam *activity diagram* pada Gambar 4.9 dan sebagian untuk *activity diagram* pada Gambar 4.17 yaitu lihat poin terdapat dalam **UserService.java** yang menyediakan *functions* untuk melakukan operasi CRUD pengguna umum. Rancangan *ER diagram* pada Gambar 4.10 diadopsikan dalam *class User.java*. Pengaturan *endpoint* yang menghubungkan *client* dengan servis terdapat dalam **UserController.java** di mana *request* dari *client* diterjemahkan menjadi *object* dari **UserRequest.java**. Layanan yang menghubungkan akses dan operasi-operasi terhadap *database* disediakan oleh **UserRepository.java**. Detail dari setiap *class* dapat dilihat pada Lampiran 4.

4.4.2 Pengembangan Servis Kualitas Udara

Ilustrasi struktur *layer project* untuk servis kualitas udara dapat dilihat pada Gambar 4.26.

Sama seperti servis pengguna, dalam servis ini hanya terdapat satu *package* servis yaitu **airqualityservice**.



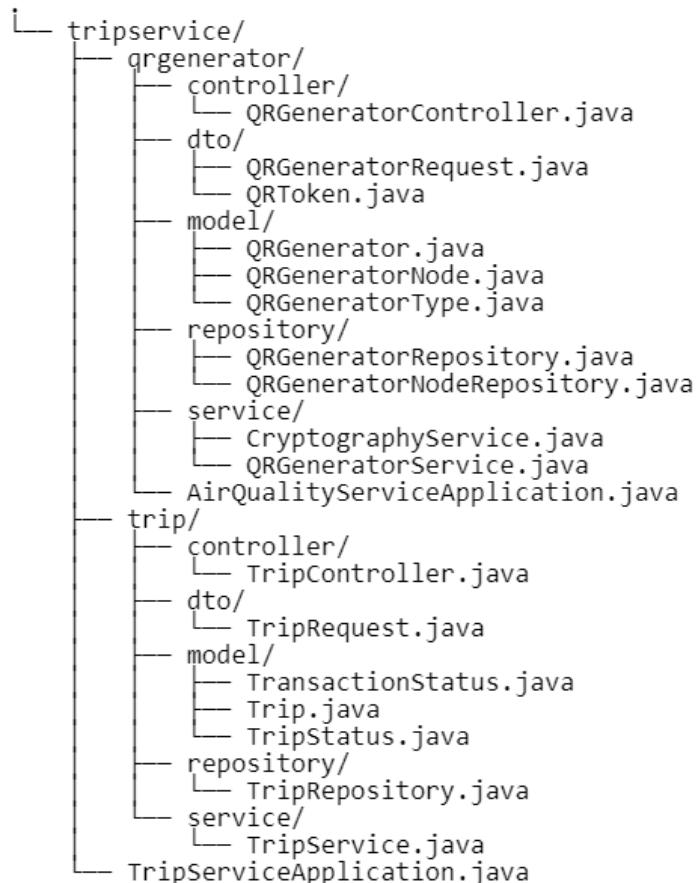
Gambar 4.26 Struktur Pohon Servis Kualitas Udara

Penerapan peran sistem dalam *activity diagram* pada Gambar 4.11 dan Gambar 4.12 untuk servis ini terdapat dalam **AirQualityService.java** yang menyediakan *functions* untuk melakukan operasi pencarian kualitas udara dan rekomendasi kegiatan. Sementara itu, **AirSensorService.java** menyediakan *functions* untuk melakukan operasi CRUD sensor udara. Rancangan *ER diagram* pada Gambar 4.13 diadopsikan dalam *class* **AirSensor.java** dan **Location.java**. Pengaturan *endpoint* terkait kualitas udara terdapat dalam **AirQualityController.java** di mana *response* untuk *client* diterjemahkan dari *schema pesan message broker* menjadi *object* dari **AirQualityResponse.java** kemudian dikirimkan dalam bentuk JSON. **AirSensorController.java** menyediakan *endpoint* terkait sensor udara di mana *request* dari *client* diterjemahkan menjadi *object* dari **AirSensorRequest.java**. Layanan yang menghubungkan akses dan operasi-operasi terhadap *database* disediakan oleh **AirQualityRepository.java** dan **AirSensorRepository.java**. Detail dari setiap *class* dapat dilihat pada Lampiran 5.

4.4.3 Pengembangan Servis Perjalanan

Ilustrasi struktur *layer project* untuk servis perjalanan dapat dilihat pada Gambar 4.27.

Terdapat dua sub-servis atau *package* dari servis perjalanan **tripservice** yaitu **qrgenerator** dan **trip** yang masing-masing menerapkan *layer* yang sama.



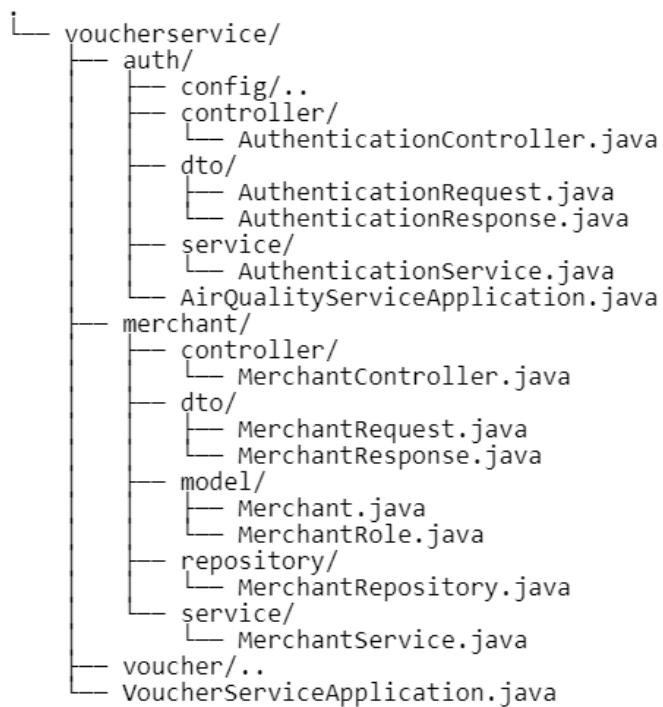
Gambar 4.27 Struktur Pohon Servis Perjalanan

Penerapan peran sistem dalam *activity diagram* pada Gambar 4.14 terdapat dalam **QRGeneratorService.java** yang menyediakan *functions* untuk melakukan operasi CRUD untuk generator *QR code* dan pembuatan *QR code*. Selain itu, penerapan Gambar 4.15 terdapat dalam **TripService.java** yang menyediakan *functions* untuk pencatatan perjalanan transportasi umum pengguna umum (*end-user*) dan lihat informasi serta riwayat perjalanan. **CryptographyService.java** menyediakan layanan *encoding* dan *decoding* kriptografi untuk keamanan token dalam *QR code*. Rancangan *ER diagram* pada Gambar 4.16 diadopsikan dalam *class* **QRGenerator.java** dan **Trip.java**. Pengaturan

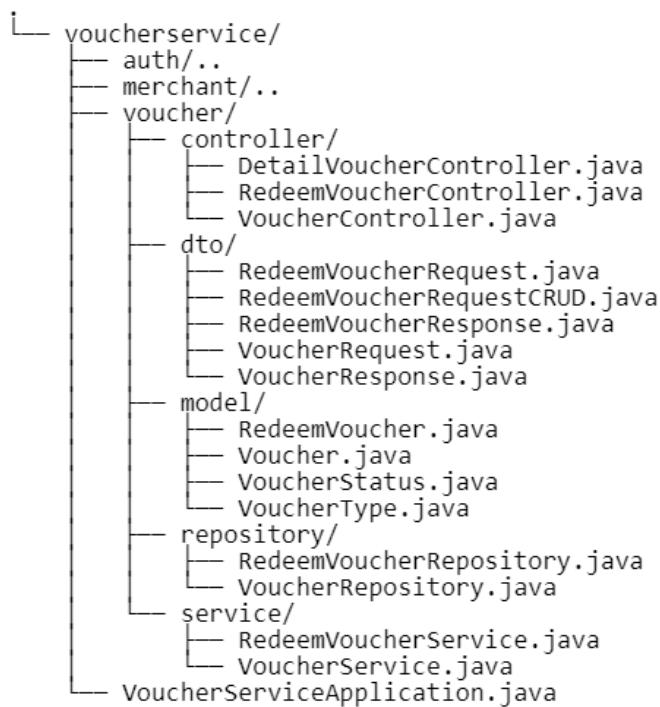
endpoint terkait generator *QR code* maupun *QR code* terdapat dalam **QRGeneratorController.java**. Dalam *file* tersebut *request* dari *client* diterjemahkan menjadi *object* dari **QRGeneratorRequest.java**. Di sisi lain, **TripController.java** menyediakan *endpoint* terkait perjalanan pengguna umum di mana *request* dari *client* diterjemahkan menjadi *object* dari **TripRequest.java**. Layanan yang menghubungkan akses dan operasi-operasi terhadap *database* disediakan oleh **QRGeneratorRepository.java**, **QRGeneratorNodeRepository.java**, dan **TripRepository.java**. Detail dari setiap *class* dapat dilihat pada Lampiran 6.

4.4.4 Pengembangan Servis Kupon dan Mitra Usaha

Ilustrasi struktur *layer project* untuk servis kupon dan mitra usaha dapat dilihat pada Gambar 4.28 dan Gambar 4.29. Terdapat tiga sub-servis dari *package* servis ini **voucherservice** yaitu **auth**, **merchant**, dan **voucher**. *Package auth* bertugas dalam autentikasi dan otorisasi mitra usaha. *Package merchant* bertugas dalam menangani segala permintaan dan *logic* terkait mitra usaha (*merchant*). *Package voucher* bertugas dalam menangani segala permintaan dan *logic* terkait penukaran poin menjadi kupon dan pengelolaan kupon oleh mitra usaha.



Gambar 4.28 Struktur Pohon Servis Kupon dan Mitra Usaha: *Auth* dan *Merchant*



Gambar 4.29 Struktur Pohon Servis Kupon dan Mitra Usaha: Voucher

Sistem keamanan autentikasi dan otorisasi diterapkan pada servis ini karena pengguna mitra usaha mengakses aplikasi berbasis web dan penerapan sistem keamanannya dapat dilakukan langsung pada servis. Selain itu, penerapan ini dapat menjadi contoh apabila seluruh servis lain terutama servis pengguna ingin menerapkan sistem keamanan akun. Sistem keamanan yang diterapkan adalah JWT (*JSON Web Token*) dengan *library* yang disediakan oleh jwt.io¹⁵. *Request* dari mitra usaha baik *login* maupun *register* diterjemahkan menjadi *object* dari **AuthenticationRequest.java** dan *response* yang memuat token JWT diterjemahkan dari bentuk *object* **AuthenticationResponse** menjadi JSON.

Penerapan peran sistem dalam *activity diagram* pada Gambar 4.20 terdapat dalam **MerchantService.java** yang menyediakan *functions* untuk melakukan operasi CRUD terhadap akun mitra usaha (*merchant*). Sementara itu, **AuthenticationService.java** menyediakan *functions* terkait otorisasi dan autentikasi mitra usaha. Pengaturan *endpoint* terkait mitra usaha terdapat dalam **MerchantController.java**. Dalam file tersebut *request* dari *client* diterjemahkan menjadi *object* dari **MerchantRequest.java** dan *response* untuk

¹⁵ <https://jwt.io/>

client diterjemahkan menjadi *object MerchantResponse.java* kemudian dikirimkan dalam bentuk JSON.

Terdapat dua *role* dari *object Merchant* yaitu *admin* dan *merchant*. Secara otomatis, ketika mitra usaha mendaftarkan akun mereka mendapatkan *role merchant*. Perbedaan mendasar antara dua *role* tersebut adalah pada pembatasan akses pada *resources*, di mana *admin* dapat mengelola semua data di servis tetapi *merchant* hanya bisa mengelola kupon dan detail kupon (*redeem voucher*) miliknya.

Rancangan *ER diagram* pada Gambar 4.23 diadopsikan dalam *class Merchant.java*, *Voucher.java*, dan *RedeemVoucher.java*. Penerapan sebagian peran sistem dalam *activity diagram* untuk lihat daftar serta detail kupon pada Gambar 4.17 terdapat dalam **VoucherService.java** dan **RedeemVoucherService.java**. Peran sistem dalam *activity diagram* pada Gambar 4.20 dan Gambar 4.21 yaitu mengelola kupon terdapat dalam **VoucherService.java**. Pengaturan *endpoint* terkait kupon terdapat dalam **VoucherController.java** di mana *request client* diubah dalam bentuk objek **VoucherRequest** untuk ditangani serta *response* untuk *client* diterjemahkan menjadi *object* dari **VoucherResponse.java** kemudian dikirimkan dalam bentuk JSON.

Di sisi lain, penerapan sebagian peran sistem dalam *activity diagram* untuk tukar kupon dan lihat riwayat pada Gambar 4.18 dan Gambar 4.19 terdapat dalam **RedeemVoucherService.java**. Pengaturan *endpoint* terkait detail kupon terdapat dalam **DetailVoucherController.java** di mana *request client* diubah dalam bentuk objek **RedeemVoucherRequestCRUD** untuk ditangani. **RedeemVoucherController.java** berisi pengaturan *endpoint* terkait penukaran poin dan kupon pengguna di mana *request client* diubah dalam bentuk objek **RedeemVoucherRequest** untuk ditangani dan *response* untuk *client* diterjemahkan menjadi *object* dari **RedeemVoucherResponse.java** kemudian dikirimkan dalam bentuk JSON. Layanan yang menghubungkan akses dan operasi-operasi terhadap *database* disediakan oleh **MerchantRepository.java**, **RedeemVoucherRepository.java**, dan **VoucherRepository.java**. Detail dari setiap *class* dapat dilihat pada Lampiran7.

4.5 Evaluasi Servis

Setelah melakukan pengembangan, servis-servis Mahoni tersebut kemudian dilakukan evaluasi untuk memastikan bahwa pengembangan tersebut menjawab pertanyaan penelitian. Evaluasi servis yang dilakukan mengikuti skenario pengujian yang tertera pada subbab 4.1.3, yaitu *unit testing*, *integration testing*, dan *end-to-end testing*.

4.5.1 Unit Testing

Pengujian *unit test* dilakukan untuk menguji apakah kode yang dibuat sesuai dengan tujuannya. Pengujian ini dilakukan kepada **package controller**, **dto**, **service**, **repository**, dan **model** yang merupakan fokus utama pengembangan dalam pekerjaan ini. Setiap servis dilakukan pengujian kemudian dilihat keberhasilan dan nilai *code coverage* yang diperoleh. *Tools* yang digunakan dalam pengujian ini adalah *plugin Coverage* oleh IntelliJ IDEA¹⁶. Penjelasan setiap kolom pada tabel hasil pengujian Tabel 4.7, Tabel 4.8, Tabel 4.9, dan Tabel 4.10 adalah sebagai berikut:

1. *Package*, persentase total keseluruhan *file* dalam *package* tersebut yang dikenakan *test*.
2. *Class*, %, persentase total jumlah *class* yang dikenakan *test* pada *package* tersebut.
3. *Method*, %, persentase total jumlah *method* yang dikenakan *test* pada *package* tersebut.
4. *Line*, %, persentase total jumlah baris *code* yang dikenakan *test* pada *package* tersebut.

Hasil pengujian *unit test* untuk servis pengguna dapat dilihat pada Tabel 4.7. Semua *unit test* sejumlah 25 *test* untuk servis ini berhasil diuji. Hasil data pengujian *unit test* untuk *class*, *method*, dan *line coverage* masing-masing adalah 100%, 95.7%, dan 95.5%. *Coverage* untuk *subpackage* selain **kafka** adalah 100% yang berarti semua *method* ataupun *line* yang perlu dilakukan pengujian terkena uji.

¹⁶ <https://www.jetbrains.com/help/idea/running-test-with-coverage.html>

Tabel 4.7 Hasil Pengujian *Unit Test* Servis Pengguna

Package	Class, %	Method, %	Line, %
com.mahoni.userservice	100% (1/1)	50% (1/2)	50% (1/2)
com.mahoni.userservice.config	100% (1/1)	100% (4/4)	100% (20/20)
com.mahoni.userservice.controller	100% (1/1)	100% (8/8)	100% (24/24)
com.mahoni.userservice.dto	100% (1/1)	100% (8/8)	100% (8/8)
com.mahoni.userservice.exception	100% (2/2)	100% (2/2)	100% (2/2)
com.mahoni.userservice.kafka	100% (4/4)	91.3% (21/23)	91.3% (84/92)
com.mahoni.userservice.model	100% (2/2)	100% (16/16)	100% (26/26)
com.mahoni.userservice.repository	100% (0/0)	100% (0/0)	100% (0/0)
com.mahoni.userservice.service	100% (1/1)	100% (7/7)	100% (28/28)
all classes	100% (13/13)	95.7% (67/70)	95.5% (193/202)

Total 25 *unit test* dibuat: 25 berhasil, 0 dilewati, 0 gagal

Hasil pengujian *unit test* untuk servis kualitas udara dapat dilihat pada Tabel 4.8. Semua *unit test* sejumlah 34 *test* untuk servis ini berhasil diuji. Hasil data pengujian unit *test* untuk *class*, *method*, dan *line coverage* masing-masing adalah 100%, 93.4%, dan 94.9%. Terdapat *package* yang tidak begitu terliput pengujian yaitu *package kafka* dengan *method* dan *line coverage* sebesar 55% karena berisi kumpulan konfigurasi untuk arsitektur *event-driven*.

Tabel 4.8 Hasil Pengujian *Unit Test* Servis Kualitas Udara

Package	Class, %	Method, %	Line, %
com.mahoni.airqualityservice	100% (1/1)	50% (1/2)	50% (1/2)
com.mahoni.airqualityservice.config	100% (1/1)	100% (4/4)	100% (20/20)
com.mahoni.airqualityservice.controller	100% (3/3)	93.8% (15/16)	98.6% (68/69)
com.mahoni.airqualityservice.dto	100% (2/2)	100% (26/26)	100% (26/26)
com.mahoni.airqualityservice.exception	100% (3/3)	100% (4/4)	100% (4/4)
com.mahoni.airqualityservice.kafka	100% (2/2)	55.6% (5/9)	55% (11/20)
com.mahoni.airqualityservice.model	100% (2/2)	100% (15/15)	100% (15/15)
com.mahoni.airqualityservice.repository	100% (0/0)	100% (0/0)	100% (0/0)
com.mahoni.airqualityservice.service	100% (2/2)	100% (15/15)	100% (61/61)
all classes	100% (16/16)	93.4% (85/91)	94.9% (206/217)

Total 34 *unit test* dibuat: 34 berhasil, 0 dilewati, 0 gagal

Tabel 4.9 Hasil Pengujian *Unit Test* Servis Perjalanan

Package	Class, %	Method, %	Line, %
com.mahoni.tripservice	100% (1/1)	50% (1/2)	50% (1/2)
com.mahoni.tripservice.common.config	100% (1/1)	100% (2/2)	100% (2/2)
com.mahoni.tripservice.qrgenerator.controller	100% (1/1)	100%	100% (34/34) (12/12)
com.mahoni.tripservice.qrgenerator.dto	100% (2/2)	100%	100% (13/13) (13/13)
com.mahoni.tripservice.qrgenerator.exception	100% (1/1)	100% (2/2)	100% (2/2)
com.mahoni.tripservice.qrgenerator.model	100% (3/3)	100%	100% (30/30) (20/20)
com.mahoni.tripservice.qrgenerator.repository	100% (0/0)	100% (0/0)	100% (0/0)
com.mahoni.tripservice.qrgenerator.service	100% (2/2)	100%	89.8% (20/20) (88/98)
com.mahoni.tripservice.trip.config	100% (1/1)	100% (4/4)	100% (20/20)
com.mahoni.tripservice.trip.controller	100% (1/1)	100% (4/4)	100% (12/12)
com.mahoni.tripservice.trip.dto	100% (1/1)	100% (6/6)	100% (6/6)
com.mahoni.tripservice.trip.kafka	100% (4/4)	92.3%	86% (37/43) (12/13)
com.mahoni.tripservice.trip.model	100% (3/3)	100%	100% (26/26) (18/18)
com.mahoni.tripservice.trip.repository	100% (0/0)	100% (0/0)	100% (0/0)
com.mahoni.tripservice.trip.service	100% (1/1)	100%	93.9% (11/11) (62/66)
all classes	100% (22/22)	98.4% (125/127)	94.1% (333/354)

Total 59 *unit test* dibuat: 59 berhasil, 0 dilewati, 0 gagal

Hasil pengujian *unit test* untuk servis perjalanan dapat dilihat pada Tabel 4.9. Semua *unit test* sejumlah 59 *test* untuk servis ini berhasil diuji. Hasil data pengujian unit *test* untuk *class*, *method*, dan *line coverage* masing-masing adalah 100%, 98.4%, dan 94.1%. Selain *subpackage kafka*, terdapat *subpackage* lain yang belum terliput *test* secara keseluruhan yaitu *subpackage trip.service* dan *qrgenerator.service*. Hal tersebut dikarenakan terdapat implementasi konsistensi arsitektur *even-driven* yang belum dikenakan *test*.

Tabel 4.10 Hasil Pengujian *Unit Test* Servis Kupon dan Mitra Usaha

Package	Class, %	Method, %	Line, %
com.mahoni.voucherservice	100% (1/1)	50% (1/2)	50% (1/2)
com.mahoni.voucherservice.auth.config	100% (4/4)	47.4% (9/19)	45.5% (30/66)
com.mahoni.voucherservice.auth.controller	100% (1/1)	100% (4/4)	100% (4/4)
com.mahoni.voucherservice.auth.dto	100% (2/2)	100% (10/10)	100% (10/10)
com.mahoni.voucherservice.auth.service	100% (1/1)	100% (4/4)	100% (18/18)
com.mahoni.voucherservice.merchant.controller	100% (1/1)	100% (8/8)	100% (25/25)
com.mahoni.voucherservice.merchant.dto	100% (2/2)	100% (14/14)	100% (14/14)
com.mahoni.voucherservice.merchant.exception	100% (2/2)	100% (3/3)	100% (3/3)
com.mahoni.voucherservice.merchant.model	100% (2/2)	100% (17/17)	100% (23/23)
com.mahoni.voucherservice.merchant.repository	100% (0/0)	100% (0/0)	100% (0/0)
com.mahoni.voucherservice.merchant.service	100% (1/1)	100% (11/11)	100% (38/38)
com.mahoni.voucherservice.voucher.config	100% (1/1)	100% (4/4)	100% (20/20)
com.mahoni.voucherservice.voucher.controller	100% (3/3)	100% (19/19)	100% (79/79)
com.mahoni.voucherservice.voucher.dto	100% (5/5)	100% (42/42)	100% (42/42)
com.mahoni.voucherservice.voucher.exception	100% (3/3)	100% (3/3)	100% (3/3)
com.mahoni.voucherservice.voucher.kafka	100% (4/4)	92.3% (12/13)	86.5% (32/37)
com.mahoni.voucherservice.voucher.model	100% (4/4)	100% (30/30)	100% (47/47)
com.mahoni.voucherservice.voucher.repository	100% (0/0)	100% (0/0)	100% (0/0)
com.mahoni.voucherservice.voucher.service	100% (2/2)	100% (24/24)	100% (123/123)
all classes	100% (39/39)	94.7%	92.4%
		(215/227)	(512/554)

Total 106 *unit test* dibuat: 106 berhasil, 0 dilewati, 0 gagal

Hasil pengujian *unit test* untuk servis kupon dan mitra usaha dapat dilihat pada Tabel 4.10. Semua *unit test* sejumlah 106 *test* untuk servis ini berhasil diuji. Hasil data pengujian *unit test* untuk *class*, *method*, dan *line coverage* masing-masing adalah 100%, 94.7%, dan 92.4%. Terdapat *package* yang tidak begitu terlilit pengujian yaitu *package auth* dengan *method* dan *line coverage* sekitar 45% karena berisi kumpulan konfigurasi keamanan Spring Boot untuk akun pengguna.

Semua pengujian *unit test* untuk setiap servis mencakup lebih dari 90% *class*, *method*, atau pun *line coverage*. Hal tersebut telah memenuhi target *coverage* minimum sejumlah 85%. Walaupun demikian, terkadang apabila menjalankan *test* ditemukan kegagalan

karena dependensi yang mengalami *timeout*. Untuk mengatasi hal tersebut, dapat dilakukan pengujian kembali.

4.5.2 Integration Testing

Pengujian *integration test* telah dijalankan bersamaan dengan *unit testing*. *Integration testing* bertujuan untuk menguji apakah servis yang telah dibuat dapat terhubung dengan dependensi yang diperlukannya. Terdapat dua dependensi yang terhubung dengan servis yaitu *message broker* Kafka dan *database* PostgreSQL. Karena berada dalam *environment testing*, implementasi dependensi tersebut menggunakan bantuan *library* Testcontainers¹⁷ yang menyediakan *containers* untuk berbagai macam *module* yang di antaranya adalah *database* dan *message broker*. Untuk setiap servis, diuji masing-masing dependensi yang digunakan tersebut. Hasil pengujian tersebut terdapat pada Tabel 4.11.

Pengujian keterhubungan Kafka dilakukan dengan melakukan *produce event* dan *consume event* dari *topic* yang bersesuaian dengan *servis*. Apabila *unit test* dalam *produce* atau *consume* berhasil, maka secara langsung dapat diketahui bahwa integrasi dependensi ke Kafka berhasil. Serupa dengan itu, pengujian keterhubungan PostgreSQL dilakukan dengan melakukan operasi *query* ke *database* seperti **insert** ataupun **select**. Apabila *unit test* untuk *query* berhasil, maka secara langsung dapat diketahui bahwa integrasi dependensi ke PostgreSQL berhasil.

Tabel 4.11 Hasil Pengujian *Integration Test* Servis-Servis Mahoni

Main Test	Unit Test	Status
UserRepositoryTest	testFindByUsername()	✓
UserEventListenerTest	givenEmbeddedKafkaBroker _whenConsumeVoucherRedeemed _thenMessageReceived() givenEmbeddedKafkaBroker _whenConsumeTrip_thenMessageReceived()	✓ ✓
UserEventProducerTest	givenEmbeddedKafkaBroker _whenSendEvent_thenMessageSent()	✓
AirSensorRepositoryTest	testFindAirSensorsByLocation() testfindAll()	✓ ✓

¹⁷ <https://www.testcontainers.org/>

Tabel 4.11 Hasil Pengujian *Integration Test* Servis-Servis Mahoni (sambungan)

Main Test	Unit Test	Status
LocationRepositoryTest	testFindAll()	✓
QRGeneratorNode-RepositoryTest	testFindAll()	✓
QRGeneratorRepositoryTest	testShortestPath()	✓
TripRepositoryTest	testFindAll()	✓
	testFindLatestActiveTripByUserId()	✓
	testFindByStatus()	✓
	testFindLatestTripByUserId()	✓
	testFindByUserId()	✓
TripEventProducerTest	givenEmbeddedKafkaBroker _whenSendEvent _thenMessageSent()	✓
TripEventListenerTest	givenEmbeddedKafkaBroker _whenConsumePoint _thenMessageReceived()	✓
RedeemVoucher-RepositoryTest	testFindAllByMerchant() testFindAllByStatus() testFindAvailableRedeemVoucherByVoucherId() testFindAllByUserId()	✓ ✓ ✓ ✓
MerchantRepositoryTest	testFindByRole() testFindByUsername()	✓ ✓
VoucherEventListenerTest	givenEmbeddedKafkaBroker _whenConsumePoint_thenMessageReceived()	✓
VoucherEventProducerTest	givenEmbeddedKafkaBroker _whenSendEvent_thenMessageSent()	✓
VoucherRepositoryTest	testFindAll()	✓

Dari 25 buah *unit test* yang digunakan dalam *integration testing*, semuanya berhasil dijalankan. *Integration testing* terhadap Kafka dilakukan ke semua servis kecuali servis kualitas udara karena tidak terdapat *producer* ataupun *consumer topic*. Sementara itu, *integration testing* terhadap *database PostgreSQL* dilakukan ke semua servis dan *database Neo4j* dilakukan ke servis perjalanan. Keberhasilan pengujian *test* menandakan bahwa implementasi servis dapat terhubung dengan dependensi yang diperlukan.

4.5.3 End-To-End Testing

End-to-end testing dilakukan untuk menguji apakah proses bisnis dapat berjalan secara semestinya di servis yang telah dibuat. Pengujian proses bisnis tersebut dapat dilakukan dengan memecah *task* menjadi satuan terkecil dalam *end-to-end testing* yaitu *thin-thread*. Penyusunan *thin-thread* dapat dilakukan dengan membuat *thin-thread tree*, yaitu *thin-thread* yang disusun dengan struktur pohon untuk menggambarkan grup atau konteks dari sekelompok *thin-thread*.

Dalam pengujian ini, penyusunan uji *thin-thread tree* dilakukan dengan bantuan tools Postman¹⁸. Postman merupakan *tools* yang dapat digunakan untuk menguji API dengan berbagai kustomisasi beserta validasi respons yang didapatkan sehingga cocok untuk digunakan dalam pengujian ini. Satu kondisi *request* HTTP Postman dapat mewakili satu *thin-thread*. Dengan begitu, *thin-thread* dapat menguji *request* yang berhasil ataupun tidak berdasarkan kondisi input yang diberikan. Kumpulan *thin-thread* tersebut dipisah berdasarkan *collection* servis ataupun *role* (khusus servis kupon dan mitra usaha). Identifikasi seluruh *thin thread* beserta hasil pengujian *end-to-end testing* tersebut terdapat pada Lampiran 8.

Terdapat prakondisi yang harus dipenuhi untuk dapat menjalankan pengujian *end-to-end* yang berhasil, terutama untuk servis perjalanan dan servis kupon dan mitra usaha. Dalam implementasinya, untuk dapat menjalankan alur dari servis perjalanan yaitu *scan trip*, maka pengguna umum harus memiliki akun terlebih dahulu sehingga harus dijalankan servis pengguna kemudian servis perjalanan. Hal serupa juga berlaku ketika ingin menjalankan pengujian alur pada servis kupon dan mitra usaha khususnya alur penukaran poin yang harus sudah terdapat akun pengguna umum kemudian akun tersebut sudah pernah mendapatkan poin dari perjalanan. Oleh karena itu, pada evaluasi umum dibuatlah skenario *end-to-end testing* yang terurut sehingga tidak perlu melakukan perulangan pengujian servis lainnya.

Dari hasil pengujian *end-to-end* seluruh *thin-thread* pada setiap servis dapat diketahui bahwa semua pengujian pada *endpoint* berhasil. Perlu diketahui bahwa pengujian ini dilakukan setelah implementasi seluruh pekerjaan penelitian ini selesai, baik untuk bab

¹⁸ <https://www.postman.com/>

4, bab 5, maupun bab 6. Hasil pengujian yang berhasil tersebut menunjukkan bahwa selain proses bisnis yang mendasari jalannya fitur-fitur sudah terimplementasi, hubungan antar servis sudah berjalan dengan baik termasuk implementasi arsitektur *event-driven* pada bab 5 dan arsitektur *big data* pada bab 6.

4.6 Kesimpulan Pengembangan Servis-Servis Mahoni

Pada bab ini telah dijelaskan tahap perancangan dan pengimplementasian servis sebagai layanan fitur aplikasi Mahoni. Kegiatan perancangan fitur dilakukan dengan *requirement gathering* untuk memastikan bahwa fitur tersebut sesuai dengan proses bisnis dan kebutuhan pengguna. Terdapat delapan tahapan yang dilakukan dalam *requirement gathering* yang dimulai dari analisis domain hingga pengelompokan kebutuhan. Hasil dari pengelompokan atau prioritisasi kebutuhan yaitu berupa sembilan fitur yang dipilih untuk dikembangkan layanannya, yaitu kualitas udara (F1), informasi dan pencarian kualitas udara (F2), rekomendasi kesehatan (F4), informasi perjalanan transportasi (F7), riwayat perjalanan (F8), jumlah poin (F9), riwayat penukaran poin (F10), daftar dan detail kupon tersedia (F11), dan daftar dan detail kupon dimiliki (F12).

Implementasi layanan fitur berupa servis dilakukan dengan mengikuti rancangan berupa *use case diagram*, *activity diagram*, dan *entity relationship diagram* yang dibuat berdasarkan translasi proses bisnis dan kebutuhan pengguna. Implementasi servis tersebut kemudian dilakukan evaluasi untuk memastikan fungsionalitas dan penerapan yang dilakukan sesuai dengan rancangan yang dibuat. Evaluasi dilakukan dengan menjalankan tes secara *bottom-up* melalui *unit testing*, *integration testing*, dan *end-to-end testing*.

Hasil evaluasi terhadap *unit testing* adalah *coverage* telah mencakup lebih dari 90% *class*, *method*, atau pun *line coverage* yang melebihi minimum target *coverage* sejumlah 85%. Hal tersebut mengindikasikan kode yang dibuat telah sesuai dengan tujuannya. Kemudian, dari hasil evaluasi terhadap *integration testing* dapat disimpulkan bahwa servis dapat terhubung dengan dependensi *database* dan *message broker* yang diperlukannya. Terakhir, seluruh *thin-thread* yang diidentifikasi pada *end-to-end testing* berhasil diuji. Dari hasil-hasil tersebut, dapat disimpulkan bahwa pengembangan servis yang dilakukan mampu melayani fitur untuk memenuhi kebutuhan pengguna.

BAB 5

ARSITEKTUR EVENT-DRIVEN MAHONI

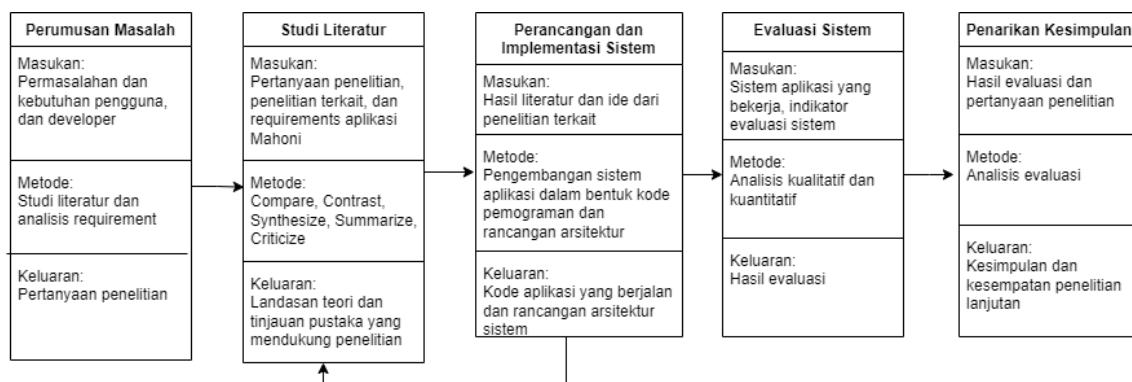
Proses penelitian dan pengembangan arsitektur *event-driven* bertujuan untuk membuat arsitektur yang paling sesuai dengan kebutuhan aplikasi Mahoni. Penelitian ini tersusun atas tiga tahap yaitu perancangan, implementasi, dan evaluasi arsitektur *event-driven*. Tahap perancangan arsitektur didasarkan kepada *requirement* yang diperoleh pada Bab 4 dan hasil studi literatur. Tahap evaluasi mengukur performa arsitektur yang sudah dibangun berdasarkan metrik *throughput* dan beberapa skenario lain. Pada bab ini juga dijelaskan keseluruhan proses dalam pembuatan arsitektur *event-driven*.

5.1 Metodologi Penelitian

Penelitian untuk menyelesaikan masalah arsitektur *event-driven* untuk aplikasi Mahoni ini menggunakan jenis penelitian eksperimental dengan pengujian berupa kuantitatif dan kualitatif. Penelitian ini menggunakan beberapa tahapan penelitian, skenario, dan matriks evaluasi.

5.1.1 Tahapan Penelitian

Tahapan penelitian yang dilakukan untuk penelitian ini memiliki tahapan yang sama seperti yang dijelaskan pada subbab 3.2.1.



Gambar 5.1 Tahapan Penelitian Arsitektur *Event-Driven*

Hal yang sedikit membedakan adalah tahap perumusan masalah didasari oleh hasil pengumpulan fitur atau *requirements* pada bab 4, sehingga masukkan untuk tahap studi

literatur adalah pertanyaan penelitian, penelitian terkait, dan *requirements* aplikasi Mahoni.

5.1.2 Skenario Pengujian

Pada arsitektur ini dilaksanakan beberapa skenario pengujian. Pengujian dibagi menjadi dua tipe, yaitu pengujian performa dan *coupling* arsitektur *event-driven*. Setiap tipe pengujian memiliki beberapa skenario yang dijelaskan lebih lanjut.

Pengujian performa dilakukan dengan menyimulasikan 200 *user* mengakses aplikasi Mahoni selama dua menit. Daftar *endpoint* tujuan dapat dilihat pada Tabel 5.1. Skenario pengujian memberikan beban pada *message broker* dan melihat dampaknya terhadap aplikasi Mahoni dengan membandingkan nilai *throughput* keseluruhan pada masing-masing skenario. Penulis membuat sebuah servis yang memproduksi *event* ke *air-quality-raw-topic* secara terus menerus dengan jumlah berbeda setiap skenario. Tabel 5.2 menunjukkan jumlah *event* yang diproduksi pada masing-masing skenario.

Tabel 5.1 Daftar *Endpoint*

No.	Method	Endpoint	Domain
1	GET	/air-quality/loc/[location]	Air Quality
2	GET	/air-sensors/[sensor_id]	Air Quality
3	POST	/api/v1/auth/register	Voucher & Merchant
4	GET	/api/v1/qr-generators/[qrgenerator_id]/generate-qr	Trip
5	GET	/api/v1/qr-generators/decode-qr	Trip
6	POST	/api/v1/redeem	Voucher & Merchant
7	POST	/api/v1/redeem/[redeem_voucher_id]	Voucher & Merchant
8	POST	/api/v1/trips	Trip
9	POST	/api/v1/users	User
10	POST	/api/v1/vouchers	Voucher & Merchant
11	POST	/api/v1/vouchers/detail	Voucher & Merchant

Tabel 5.2 Skenario Pengujian Performa

No.	Maksimum Concurrent User	Event yang diproduksi per Detik
1	200	100
2	200	1000
3	200	10000

Selanjutnya, pengujian *coupling* dilakukan dengan mengintegrasikan sebuah servis baru saat seluruh aplikasi Mahoni sedang berjalan. Pengujian ini bersifat kuantitatif karena tidak ada metrik yang digunakan, melainkan dievaluasi berdasarkan informasi yang dibutuhkan dan demonstrasi untuk membuktikan tahap apa saja yang diperlukan agar servis baru dapat terintegrasi ke sistem. Servis baru yang ditambahkan diharapkan dapat berjalan dengan seminim mungkin informasi yang diperlukan, dan tanpa mengganggu keberadaan servis lain dalam sistem. Servis yang ditambahkan adalah servis loyalitas, yang bertugas untuk menghitung poin loyalitas sebuah *user* yang dapat dilihat di Persamaan 5.1

$$\begin{aligned} \text{total poin yang didapatkan} &+ (1.5 \\ &\ast \text{ total poin yang dipakai}) \end{aligned} \quad (5.1)$$

Perhitungan poin dilakukan dengan mengonsumsi *user-point-topic* sebagai sumber data dan memastikan bahwa data seluruh *user* diperoleh hanya dari *topic* tersebut.

5.1.3 Matriks Evaluasi

Skenario pengujian performa memiliki beberapa matriks evaluasi sebagai pengukuran keberhasilan arsitektur *event-driven*. Pengujian performa membandingkan *throughput* dan *response time* antar skenario. Matriks dalam pengukuran *throughput* di sini adalah jumlah total data dan data per detik. Data yang diukur berupa *event* di Kafka dan *request user*. Pengukuran *response time* dapat membantu melihat dampak yang terjadi pada servis apabila *message broker* sedang mengalami beban yang tinggi. Dengan mengukur kedua data tersebut dapat memperlihatkan seberapa besar *throughput* yang diperoleh menggunakan arsitektur ini.

5.2 Perancangan Arsitektur *Event-Driven*

Perancangan arsitektur *event-driven* dilakukan untuk membuat arsitektur yang sesuai dengan kebutuhan aplikasi Mahoni dan bisa memberikan performa atau *throughput* yang baik. Tahap ini dimulai dengan menentukan *framework event-driven* yang digunakan. Pemilihan *framework* didasarkan pada kebutuhan aplikasi untuk memproses *event* dalam jumlah yang banyak dan didukung oleh penelitian sebelumnya yang membahas terkait performa dan kemudahan untuk integrasi dengan sistem *big data*.

Selanjutnya, penulis merancang arsitektur *message broker* dan *topic* yang diperlukan. Kemudian dibuat juga *schema* untuk *event* dari masing-masing *domain*. *Schema* ini nantinya dapat digunakan sebagai dokumentasi *event* dalam sistem Mahoni untuk mempermudah proses pengembangan. Alur komunikasi antar *domain* disesuaikan dengan *activity diagram* yang telah dibuat. Dijelaskan pula arsitektur *change data capture* untuk integrasi dengan komponen *big data*. Terakhir, diperlihatkan hasil arsitektur secara keseluruhan ketika seluruh komponen *event-driven* dan *change data capture* sudah diintegrasikan.

5.2.1 Pemilihan *Framework*

Pada bagian ini dijelaskan alasan pemilihan suatu *framework*. Seperti yang dijelaskan pada tahap penelitian, proses pemilihan suatu teknologi didasari pada studi literatur dan kebutuhan aplikasi Mahoni. Terdapat beberapa teknologi yang perlu dipilih, yaitu *message broker* dan *change data capture*. Pemilihan *framework* web telah dibahas pada bab 4 tentang pengembangan servis aplikasi Mahoni sehingga tidak dibahas di sini.

5.2.1.1 *Message Broker*

Pemilihan *message broker* dibatasi kepada dua pilihan yang paling populer yaitu RabbitMQ dan Kafka. Terdapat perbedaan fitur yang ditawarkan oleh kedua pilihan tersebut. RabbitMQ merupakan *message broker* berbasis *queue*, sedangkan Kafka berbasis *log*. Terdapat penelitian yang dilakukan oleh Dobbelaere dan Esmaili pada tahun 2017 yang dapat membantu menjawab pertanyaan *message broker* apa yang sesuai untuk dipilih. Penelitian tersebut menghasilkan Tabel 5.3 (Dobbelaere & Esmaili, 2017) yang dapat disesuaikan dengan kebutuhan aplikasi dan solusi yang sesuai.

Pada penelitian tersebut juga disebutkan beberapa fitur yang unik pada masing-masing teknologi. Fitur unik pada Kafka adalah kemampuan untuk menyimpan dalam jangka panjang, melakukan *message replay*, *platform Kafka Connect*, dan *log compaction*. Di sisi lain, fitur unik pada RabbitMQ adalah protokol standar (AMQP), mendukung banyak protokol seperti MQTT (sangat populer pada IoT), *publisher flow control*, dan *queue size limits*.

Tabel 5.3 RabbitMQ dan/atau Kafka

Predictable latency?	Complex routing?	Long term storage?	Very large throughput per topic?	Packed order important?	Dynamic elasticity behavior?	System throughput?	At least once?	High availability?	
N	N	*	*	N	N	XL	N	N	Kafka with multiple partitions
N	N	*	*	N	N	XL	Y	Y	Kafka with replication and multiple partitions
N	N	*	*	Y	N	L	N	N	Single partition Kafka
N	N	*	*	Y	N	L	Y	Y	Single partition Kafka with replication
*	*	N	N	*	*	L	*	N	RabbitMQ
*	*	N	N	*	*	L	*	Y	RabbitMQ with queue replication
*	*	Y	N	*	*	L	*	*	RabbitMQ with Kafka long term storage
N	Y	*	*	N	N	XL	N	*	Kafka with selected RabbitMQ routing

Sumber: Dobbelaere & Esmaili (2017), telah diolah kembali

Aplikasi Mahoni merupakan aplikasi kota cerdas yang menggunakan sensor kualitas udara untuk perhitungan poin. Terdapat pula kebutuhan untuk melakukan integrasi dengan arsitektur *big data* untuk *real-time monitoring* menggunakan *stream processing*. Salah satu kebutuhan penting dalam aplikasi kota cerdas adalah integrasi yang mudah terhadap komponen baru dalam sistem. Untuk hal tersebut diperlukan sebuah *message broker* yang bisa mempertahankan *event* dalam jangka waktu panjang. Sistem kota cerdas juga memiliki banyak sekali *event* apabila seluruh sensor telah berjalan. Kedua hal

tersebut membuat Kafka lebih cocok untuk diimplementasikan sebagai *message broker* pada penelitian ini. Meskipun demikian, terdapat tantangan tersendiri untuk menyimpan data yang perlu urutan dan korelasi pada Kafka dengan banyak partisi. Salah satu contoh adalah data historis sebuah *user*. Penulis menjelaskan solusi mengatasi hal tersebut di bagian Rancangan Arsitektur *Event-Driven*.

5.2.1.2 Change Data Capture

Salah satu kebutuhan arsitektur *big data* adalah *stream processing*. Untuk melakukan hal tersebut diperlukan data yang digunakan untuk proses *enrichment* (dijelaskan lebih lanjut pada Bab 6). Secara teori arsitektur *big data* bisa saja langsung terhubung dengan *database* di masing-masing servis saat memerlukan data terkait, namun hal tersebut dapat mempengaruhi aplikasi secara keseluruhan karena performa *database* akan menurun. Untuk mengatasi hal tersebut, penulis memutuskan untuk menggunakan *change data capture* (CDC) untuk mereplikasi data yang dibutuhkan ke *database* lain tanpa mengurangi performa *database*.

Sudah terdapat beberapa alat yang dapat digunakan untuk mempermudah proses CDC menggunakan Kafka. Kafka sendiri menyediakan Kafka Connect yang dapat mempermudah integrasi antar Kafka dengan sistem lain. Selain itu, terdapat pula alat lain bernama Debezium yang berfungsi untuk menghubungkan *database* ke Kafka secara langsung. Debezium sendiri sudah mendukung sumber data dari beberapa *database* termasuk PostgreSQL. Di sisi lain, Kafka Connect juga sudah memiliki implementasi Cassandra *sink connector* dan Google yang tersedia secara gratis. Hal ini berarti seluruh proses CDC menggunakan Debezium dan Kafka Connect sudah dapat mengatasi kebutuhan perpindahan data dari sumber *database* menuju tujuan akhir *database*. Oleh karena itu, penulis memilih untuk menggunakan kedua alat ini untuk mempermudah proses CDC.

5.2.2 Rancangan Arsitektur *Event-Driven*

Pada bagian ini dibahas bagaimana perancangan arsitektur *event-driven* untuk aplikasi Mahoni. Penjelasan dimulai dengan bagaimana bentuk *event* dan *topic* tujuannya di Kafka. Selanjutnya dibahas pula beberapa kasus tentang konsistensi serta rencana untuk mengatasinya. Terdapat pula pembahasan tentang *change data capture* dan bagaimana

arsitektur tersebut dibuat. Terakhir, dijelaskan seluruh komponen arsitektur secara keseluruhan.

5.2.2.1 Kafka Cluster

Kafka *cluster* yang digunakan memanfaatkan sebuah *node Zookeeper* dan tiga Kafka *broker*. Jumlah broker tersebut diambil dari diskusi forum oleh para developer yang sudah pernah menggunakan Kafka pada *production environment* dan beberapa blog atau dokumentasi di internet. Sebuah blog yang ditulis oleh Douglas Hellinger pada tahun 2022¹⁹ menyatakan bahwa jumlah minimum Kafka *broker* adalah tiga untuk mengantisipasi adanya *broker* yang *crash*. Dengan mengatur jumlah *broker* bernilai tiga dan *replication factor* bernilai dua, setiap partisi pada Kafka dipastikan memiliki minimal dua *in-sync replica*. Nilai ini juga sesuai dengan dokumentasi penyedia *real-time data analytics* Cloudera²⁰ yang memberikan rekomendasi tiga Kafka *broker* untuk dapat mendapatkan performa setara 225MB/detik.

Selain dari itu, dengan membuat jumlah *broker* lebih dari satu, penulis dapat memanfaatkan sifat *horizontal-scaling* dengan membuat *topic* yang memiliki banyak partisi. Masing-masing partisi kemudian dapat dikonsumsi oleh *consumer* yang berbeda. Pembahasan mengenai rancangan *topic* dibahas pada subbab selanjutnya.

5.2.2.2 Kafka Topic

Hal selanjutnya yang perlu dilakukan setelah menentukan *Kafka cluster* adalah pembuatan *topic* beserta konfigurasinya. Pada buku *Kafka: The Definitive Guide* dijelaskan bahwa jumlah partisi yang dibutuhkan untuk setiap *topic* bergantung dengan kebutuhan aplikasi. Namun, buku tersebut juga memberikan saran untuk menentukan jumlah partisi dengan melihat potensi aplikasi di masa depan karena proses penambahan jumlah partisi saat aplikasi sudah berjalan akan sulit. Melihat dari saran tersebut, maka penulis memilih untuk memaksimalkan seluruh partisi di seluruh *topic* dengan banyaknya jumlah *broker* Kafka dengan asumsi bahwa akan ada penambahan berbagai macam sensor di aplikasi kota cerdas Mahoni ke depannya.

¹⁹ <https://learnk8s.io/kafka-ha-kubernetes>

²⁰ https://docs.cloudera.com/HDPDocuments/HDF3/HDF-3.1.0/bk_planning-your-deployment/content/ch_hardware-sizing.html

Hampir seluruh *topic* yang dibuat memiliki *retention type* bernilai *time-based* dengan pengecualian pada *air-quality-compacted-topic* dan *user-point-compacted-topic*. Alasan terdapat perbedaan pada kedua *topic* ini adalah karena data yang disimpan pada *topic* tersebut sering diakses oleh domain lain dan dapat memanfaatkan *compaction* untuk hanya mendapatkan nilai terkini dari data. Selain itu, KTable juga dapat dimanfaatkan untuk mempermudah konsumsi dari *compacted topic* dan membuatnya dapat diakses layaknya *key-value database*. Fitur ini juga membuat *flow* menjadi lebih simpel karena tidak perlu secara *asynchronous* meminta data pada *domain* lain. Tabel 5.4 memperlihatkan daftar *topic* di aplikasi Mahoni dan konfigurasinya.

Tabel 5.4 Daftar Kafka *Topic* Aplikasi Mahoni beserta Konfigurasinya

Nama Topic	Domain	Retention	Jumlah	Jumlah	Nama Schema	Dikonsumsi oleh
		Type	Partisi	Replikasi		
user-point-topic	User Service	<i>Time-based</i>	3	2	user-point-v1	Trip Service
user-point-compacted-topic	User Service	<i>Key-based</i>	3	2	user-point-table-v1	Voucher Service
air-quality-raw-topic	Air Quality Service	<i>Time-based</i>	3	2	air-quality-raw-v1	Flink (<i>stream processor</i>)
air-quality-processed-topic	Air Quality Service	<i>Time-based</i>	3	2	air-quality-processed-v1	Air Quality Service
air-quality-compacted-topic	Air Quality Service	<i>Key-based</i>	3	2	air-quality-compacted-v1	Trip Service
trip-topic	Trip Service	<i>Time-based</i>	3	2	trip-event-v1	User Service
voucher-redeemed-topic	Merchant & Voucher Service	<i>Time-based</i>	3	2	voucher-redeemed-event-v1	User Service

Ada catatan penting yang perlu diingat yaitu perlu perlakuan khusus untuk beberapa *topic* yang menyimpan data historis agar setiap *event* di dalam partisi tetap koheren dan terurut

jika dikonsumsi oleh *consumer* yang berbeda. *Topic* yang dimaksud adalah *trip-topic*, *user-point-topic*, dan *voucher-redeemed-topic*. Ketiga *topic* ini menyimpan data historis sebuah *user*, sehingga setiap *event* yang berhubungan dengan suatu *user* harus disimpan di partisi yang sama. Solusi untuk menyelesaikan masalah ini adalah dengan menentukan secara manual partisi yang sesuai untuk setiap *event* berdasarkan “*userId*” sebagai acuan perhitungan. Akan dijelaskan lebih lanjut proses implementasinya di subbab Implementasi Arsitektur *Event-Driven*.

5.2.2.3 Domain Event dan Schema

Hasil penelitian pada bab 4 mengategorikan *requirements* menjadi empat *domain* berbeda yaitu *air quality*, *merchant & voucher*, *user*, dan *trip*. Untuk memastikan adanya *boundary context*, dibuat juga *schema event* pada masing-masing *domain*. Hanya *domain* yang memiliki *event* tersebutlah yang boleh mengirim *event* dan mengubah *schema*-nya. Seluruh *schema event* dapat dilihat di Lampiran 9.

Melihat bahwa bentuk *event* cukup kompleks, maka penulis memilih untuk menggunakan *schema registry* agar setiap *event* konsisten. Komponen ini juga dapat digunakan sebagai dokumentasi *event* untuk mempermudah proses integrasi dengan komponen baru yang ingin menerima *event* dari masing-masing *topic*. *Schema registry* yang diimplementasikan terdiri dari sebuah *node* untuk satu Kafka *cluster*.

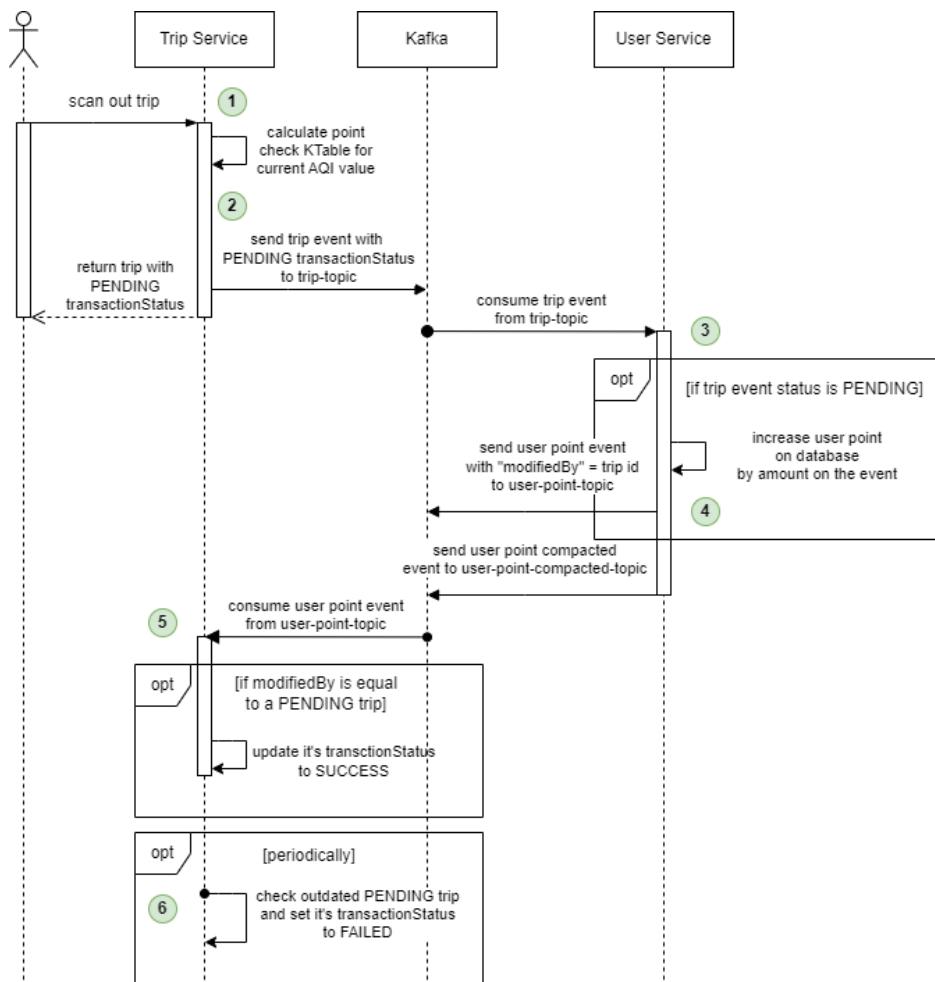
5.2.2.4 Servis Web dan Database

Setiap servis dibangun dengan *framework* Spring Boot dan PostgreSQL *database* dengan bantuan *library* Spring Data. Untuk servis *trip* diperlukan *graph database* untuk memodelkan rute transportasi umum. *Database* yang dipilih adalah Neo4j karena sudah terdapat dukungan resmi dari Spring Data sehingga mempermudah proses integrasi.

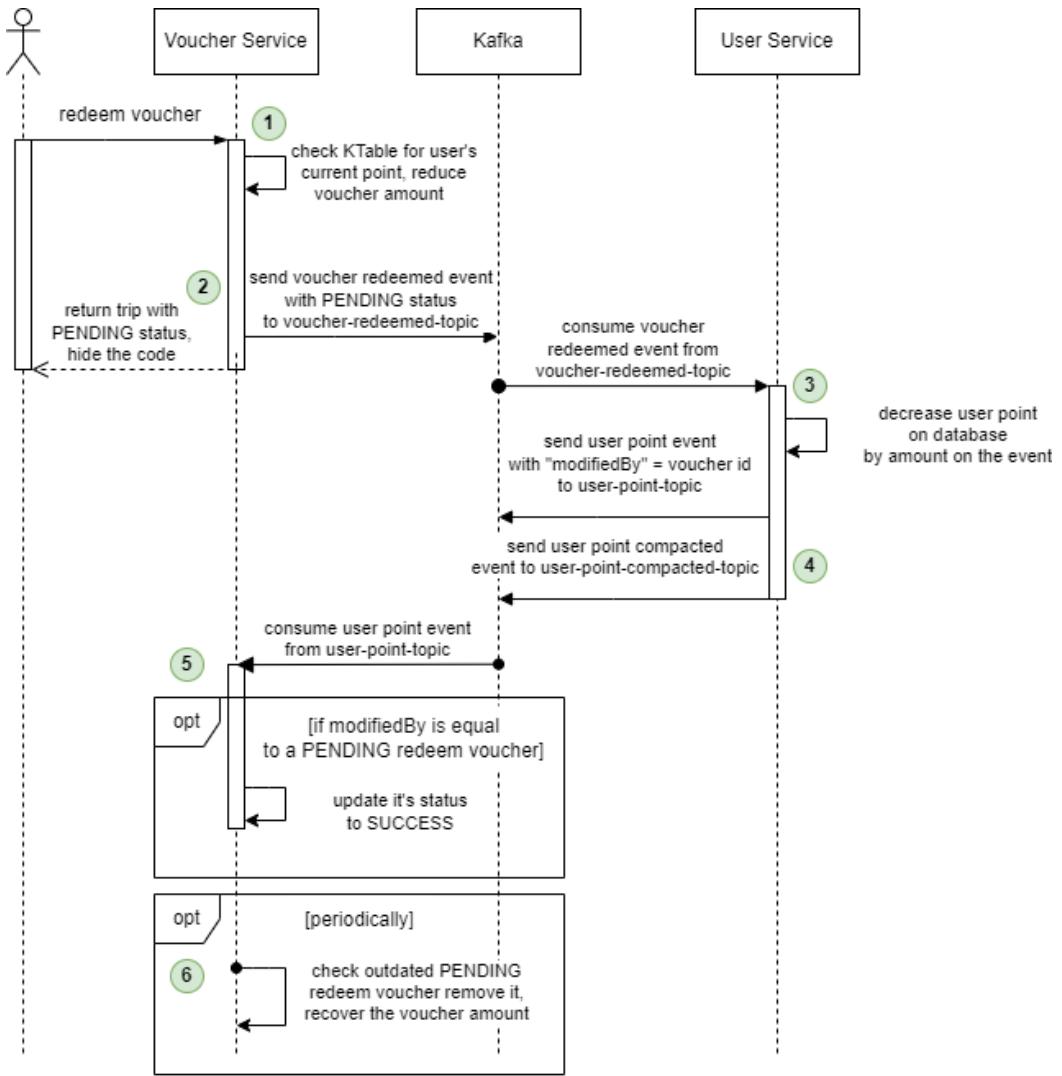
Untuk berkomunikasi satu sama lain, seluruh servis menggunakan Kafka sebagai *message broker* dengan bantuan *library* SpringKafka. Selain itu untuk servis yang menggunakan KTable juga menggunakan KafkaStreams sebagai *library* tambahan. Konfigurasi *library* SpringKafka dan KafkaStreams dijelaskan pada subbab Implementasi Arsitektur *Event-Driven*.

5.2.2.5 Konsistensi

Penggunaan *event-driven microservice* sebagai arsitektur Mahoni memiliki beberapa tantangan tersendiri, terutama pada sifat konsistensi dan proses transaksi. Terdapat dua *user flow* yang termasuk ke dalam ini yaitu *trip flow*, dan *redeem voucher flow*. *Trip flow* memerlukan kolaborasi antara servis *trip* dan *user* untuk mengubah poin *user* berdasarkan hasil perhitungan ketika sebuah trip selesai dilakukan, sedangkan *redeem voucher flow* memerlukan kolaborasi antara servis *merchant & voucher* dan *user* untuk melihat jumlah poin *user* saat ini dan mengubahnya ketika melakukan *redeem*. Kedua *flow* ini memerlukan adanya konsistensi pada nilai poin *user* untuk memastikan bahwa poin berubah jika dan hanya jika *trip selesai* atau sebuah voucher berhasil di-redeem. Oleh karena itu, dibuat *flow* yang memastikan bahwa kedua proses ini bersifat *eventually consistent*.



Gambar 5.2 *Trip Flow* untuk Memastikan Konsistensi Nilai Poin *User*



Gambar 5.3 *Redeem Voucher Flow* untuk Memastikan Konsistensi Nilai Poin User

Penjelasan *trip flow* pada Gambar 5.2 adalah sebagai berikut:

1. Setiap kali *user scan out* akan memulai proses perhitungan poin berdasarkan waktu dan AQI dari *trip* yang dilakukan. Servis *trip* mengambil nilai AQI dari KTable *air-quality-state-store*.
2. Servis *trip* mengirimkan kembali respons ke *user* tentang *trip* dengan *transactionStatus* bernilai PENDING. Kemudian mengirim *event* ke *trip-topic*.
3. Servis *user* mendengarkan *event* pada *trip-topic* dan mengecek apabila ada *transactionStatus* yang bernilai PENDING. Bila ada lakukan perubahan poin di *database*.

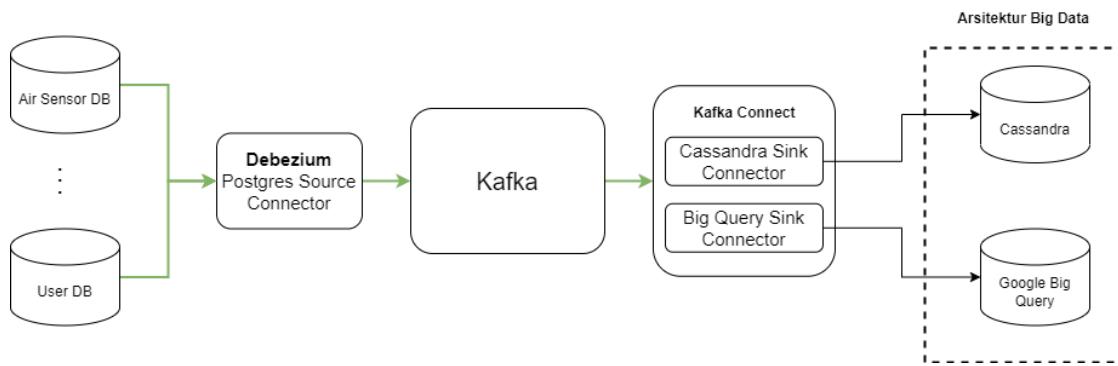
4. Servis *user* mengirimkan *event* tentang perubahan poin dengan nilai *modifiedBy* sesuai *id trip* ke *user-point-topic*. Kemudian mengirimkan *event* ke *user-point-compacted-topic* dengan *userId* sebagai *key*.
5. Servis *trip* mendengarkan *user-point-topic* dan mengecek apabila ada *modifiedBy* yang sama dengan *id* sebuah PENDING *trip*. Apabila *trip* masih berlaku, ubah *transactionStatus* *trip* tersebut menjadi SUCCESS.
6. Secara berkala akan mengecek seluruh PENDING *trip*, apabila sudah melewati batas expirasi, maka *transactionStatus* diubah menjadi FAILED.

Penjelasan *redeem voucher flow* pada Gambar 5.3 adalah sebagai berikut:

1. Setiap kali *user redeem voucher* akan servis *voucher* akan mengambil poin *user* saat ini dari KTable *user-point-state-store* kemudian mengurangi jumlah *voucher*.
2. Servis *voucher* mengirimkan kembali respons ke *user* tentang *reedeem voucher* dengan status bernilai PENDING dan menyembunyikan kodennya. Kemudian mengirim *event* ke *voucher-redeemed-topic*.
3. Servis *user* mendengarkan *event* pada *voucher-redeemed-topic* dan melakukan perubahan poin di *database*.
4. Servis *user* mengirimkan *event* tentang perubahan poin dengan nilai *modifiedBy* sesuai *id redeem voucher* ke *user-point-topic*. Kemudian mengirimkan *event* ke *user-point-compacted-topic* dengan *userId* sebagai *key*.
5. Servis *voucher* mendengarkan *user-point-topic* dan mengecek apabila ada *modifiedBy* yang sama dengan *id* sebuah PENDING *redeem voucher*. Apabila masih berlaku, ubah statusnya menjadi SUCCESS.
6. Secara berkala akan mengecek seluruh PENDING *redeem voucher*, apabila sudah melewati batas expirasi, maka hapus *redeem voucher* tersebut dan kembalikan jumlah *voucher*.

5.2.2.6 Change Data Capture

Seperti yang sudah dijelaskan sebelumnya, penulis memilih untuk menggunakan Debezium dan Kafka Connect untuk implementasi CDC. Arsitektur CDC yang diimplementasikan sesuai dengan gambar arsitektur yang tertera pada dokumentasi Debezium²¹ seperti pada Gambar 5.4 (Debezium Architectur, n.d.).



Gambar 5.4 Arsitektur Change Data Capture dengan Debezium dan Kafka Connect

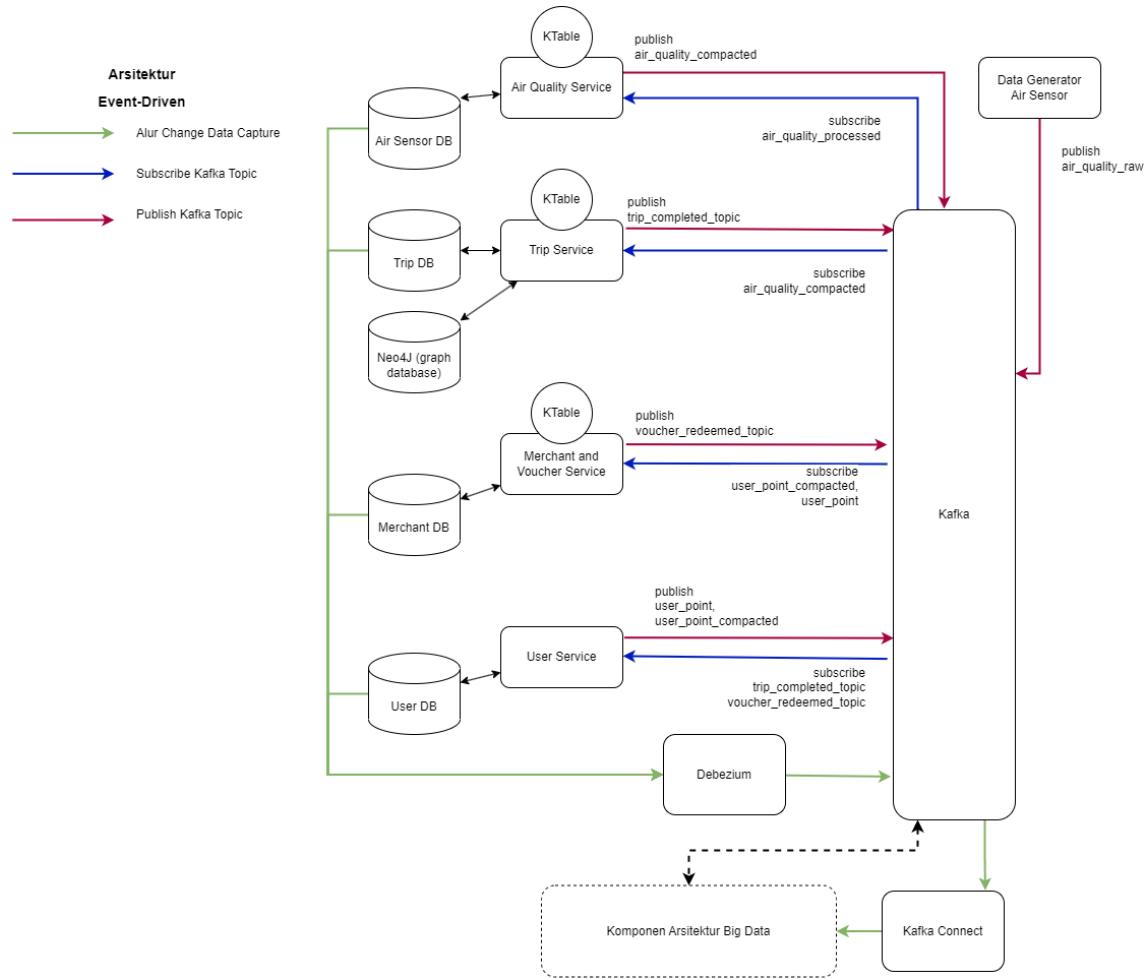
Sumber: Debezium Architecture (n.d.), telah diolah kembali

Debezium digunakan untuk membaca WAL pada *database* dan mengirimnya menjadi *event* Kafka. Selanjutnya, Kafka Connect membaca *event* tersebut dan memindahkannya ke Cassandra dan *data warehouse*. Penggunaan Cassandra dan Google Big Query sebagai *database* tujuan dibahas lebih lanjut pada bab 6 tentang arsitektur *big data*.

5.2.2.7 Arsitektur *Event-Driven*

Gambar 5.5 merupakan hasil perancangan arsitektur *event-driven* ketika seluruh komponen sudah diintegrasikan. Pada gambar tersebut terlihat alur dari proses *change data capture* yang dimulai dari *database* pada masing-masing domain kemudian berakhir di komponen arsitektur *big data*. Proses ini dibantu dengan Debezium, Kafka, dan Kafka Connect. Proses CDC ini memanfaatkan Kafka sebagai *log* untuk menyimpan *event* perubahan pada *database* dan menyalirkannya ke *database* lain.

²¹ <https://debezium.io/documentation/reference/stable/architecture.html>



Gambar 5.5 Arsitektur *event-driven* aplikasi Mahoni

Gambar tersebut juga menjelaskan alur komunikasi antar servis dengan proses *subscribe* dan *publish* sesuai dengan proses bisnis yang diperlukan. Terdapat pula lokasi di mana KTable diimplementasikan untuk mengonsumsi *compacted topic* sebagai bagian dari proses komunikasi antar servis pada sistem. Terlihat bahwa Kafka digunakan sebagai perantara untuk berkomunikasi, tidak hanya antar sesama servis Mahoni, melainkan juga antara komponen *big data* dan komponen arsitektur *event-driven*.

5.3 Implementasi Arsitektur *Event-Driven*

Implementasi arsitektur dapat dimulai setelah perancangan selesai. Melihat dari gambar arsitektur *event-driven* aplikasi Mahoni, terdapat banyak komponen yang perlu diimplementasikan. Komponen-komponen tersebut dapat dikategorikan menjadi servis, *message broker*, dan *change data capture*.

5.3.1 Infrastruktur pada Docker Virtual Machine

Komponen Kafka *cluster*, *schema registry*, dan *change data capture* dikembangkan menggunakan Docker *image*. Seluruh komponen yang dijelaskan pada bagian ini di-deploy pada Google Cloud Platform *compute engine* bertipe e2-highmem-2 dengan spesifikasi 2 vCPU, 16 GB *memory*, dan 20 GB *storage*. Alasan seluruh komponen tersebut diletakkan pada satu tempat yang sama adalah keterbatasan *resource* Google Cloud Platform yang dimiliki penulis dan untuk memudahkan proses *deployment*. Kode *docker-compose* lengkap yang digunakan untuk menjalankan seluruh komponen ini dapat dilihat di *repository*²². Penulis juga menggunakan *template compute engine* yang telah dipasang Docker di dalamnya, *template* yang digunakan dapat diakses pada *marketplace*²³.

5.3.1.1 Kafka Cluster dan Schema Registry

Implementasi Kafka *cluster* dan *schema registry* dilakukan menggunakan *docker-compose* dengan *image* resmi yang dikelola oleh Confluent Inc. Lampiran 13 memperlihatkan konfigurasi *docker-compose* yang digunakan untuk membuat Kafka *cluster* dan *schema registry*. Karena seluruh komponen berada pada satu *instance* yang sama, maka diperlukan pengaturan *port* agar tidak terjadi bentrok. Zookeeper terletak pada *port* 2181, Kafka *broker* pada *port* 9092-9094, dan *schema registry* pada *port* 8081. Seluruh *broker* terhubung satu sama lain dengan bantuan Zookeeper yang dikonfigurasikan dengan menaruh *url*-nya pada variabel **KAFKA_ZOOKEEPER_CONNECT**. *Schema registry* juga terhubung dengan seluruh *cluster* karena bantuan dari Zookeeper.

²² <https://github.com/Mahoni-Smart-City/mahoni-core/blob/main/deployment/docker-compose/docker-compose.kafka.yml>

²³ [https://console.cloud.google.com/compute/instancesAdd\(cameo:product/cloud-infrastructure-services/docker-ubuntu-20\)?authuser=1&project=famous-buckeye-386102](https://console.cloud.google.com/compute/instancesAdd(cameo:product/cloud-infrastructure-services/docker-ubuntu-20)?authuser=1&project=famous-buckeye-386102)

Broker ID	Disk usage	Partitions skew	Leaders	Leader skew	Online partitions	Port	Host
1	228 MB, 114 segment(s)	-1.70%	53	-3.60%	114	29092	kafka
2	233 MB, 115 segment(s)	-0.90%	53	-3.60%	115	29092	kafka-2
3	230 MB, 119 segment(s)	2.60%	59	7.30%	119	29092	kafka-3

Gambar 5.6 Tampilan Kafka UI untuk memonitor Kafka *cluster*

Selain itu, penulis menggunakan modul Kafka UI untuk memonitor kondisi Kafka *cluster*. Modul ini sangat berguna untuk membantu melihat daftar *topic*, *consumer*, *schema*, dan status *broker*. Modul ini juga bisa melihat kondisi masing-masing broker dan pembagian partisi untuk setiap topik. Gambar 5.6 memperlihatkan tampilan Kafka UI.

5.3.1.2 *Change Data Capture*

Terdapat beberapa hal yang perlu dikonfigurasi dalam melakukan *change data capture*. Perlu dilakukan konfigurasi WAL pada PostgreSQL terlebih dahulu, kemudian konfigurasi Debezium untuk menghubungkan PostgreSQL dengan Kafka, dan terakhir konfigurasi Kafka Connect untuk menghubungkan Kafka dengan Cassandra dan *data warehouse*.

5.3.1.2.1 Konfigurasi Replikasi dan WAL pada PostgreSQL

Mengikuti dokumentasi Debezium²⁴, hal pertama yang perlu dilakukan adalah melakukan konfigurasi pada *database* PostgreSQL. Setelah versi PostgreSQL 9.4, *plugin logical decoding* sudah tersedia secara langsung dan dapat langsung digunakan.

```
# Replication
wal_level = logical
```

Kode 5.1 Konfigurasi WAL pada PostgreSQL

²⁴ <https://debezium.io/documentation/reference/stable/connectors/postgresql.html>

```
ALTER USER existing_user WITH REPLICATION;
```

Kode 5.2 Perintah Mengubah *Role User* pada PostgreSQL

Kode 5.1 berisi konfigurasi WAL untuk PostgreSQL sebelum *database* dibuat. Penulis menggunakan PostgreSQL yang disediakan oleh Google Cloud Platform, sehingga konfigurasi yang perlu dilakukan adalah menambahkan *flag* **cloud_sql.enable_pglogical** dan **cloud_sql.logical_decoding** saat pembuatan *instance* PostgreSQL. Selanjutnya perlu diatur juga *role user* untuk *replication*. Kode perintah untuk mengubah *role* terdapat pada Kode 5.2.

5.3.1.2.2 Debezium dan *Source Connector*

Debezium dapat dijalankan di Docker menggunakan *image* resmi. Karena seluruh event diserialisasi dengan Avro, maka perlu dicantumkan juga *class* **AvroConverter** pada **KEY_CONVERTER** dan **VALUE_CONVERTER**, sedangkan variabel **SCHEMA_REGISTRY_URL** diarahkan ke *schema registry* di mana seluruh *schema* disimpan. Konfigurasi Docker *container* untuk debezium dapat dilihat di Kode 5.3.

```
services:
  debezium:
    restart: always
    image: debezium/connect:1.9
    environment:
      BOOTSTRAP_SERVERS: PLAINTEXT://kafka:29092,PLAINTEXT://kafka-2:29092,PLAINTEXT://kafka-3:29092
      GROUP_ID: debezium-group-id
      CONFIG_STORAGE_TOPIC: connect_configs
      OFFSET_STORAGE_TOPIC: connect_offsets
      KEY_CONVERTER: io.confluent.connect.avro.AvroConverter
      VALUE_CONVERTER: io.confluent.connect.avro.AvroConverter
      CONNECT_KEY_CONVERTER_SCHEMA_REGISTRY_URL: http://kafka-schema-registry:8081
      CONNECT_VALUE_CONVERTER_SCHEMA_REGISTRY_URL: http://kafka-schema-registry:8081
    ports:
      - "8083:8083"
```

Kode 5.3 Konfigurasi *docker-compose* Debezium

Setelah Debezium berhasil berjalan, maka perlu dibuat konfigurasi *connector* untuk menghubungkannya dengan *database*. Setiap koneksi Debezium dan *database* dapat dibuat dengan mengirimkan POST *request* ke *endpoint* <HOST>:8083/connectors

dengan JSON *payload* yang berisi konfigurasi *connector*. *User* pada konfigurasi *connector* harus memiliki *role* **REPLICATION** seperti yang sudah diatur pada subbab sebelumnya. Pada Kode 5.4 terdapat contoh konfigurasi *connector* untuk *database* pada servis *user*. Seluruh konfigurasi *connector* yang lengkap dapat dilihat di *repository*²⁵. Jenis *connector* yang digunakan adalah **PostgresConnector** sesuai dengan *database*. Variabel **key.converter** dan **value.converter** disesuaikan dengan konfigurasi *container* Debezium yaitu dengan menggunakan *class* **AvroConverter**. Kemudian variabel **publication.autocreate.mode** diset menjadi “*all_tables*” untuk menangkap perubahan pada seluruh tabel di dalam *database*. Penulis juga menggunakan *class* **ExtractNewRecordState** agar setiap *event* hanya menyimpan nilai akhir dari perubahan pada *database* demi kemudahan untuk integrasi *sink connector*.

```
{
  "name": "userdb-connector",
  "config": {
    "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
    "plugin.name": "pgoutput",
    "database.hostname": "34.128.102.22",
    "database.port": "5432",
    "database.user": "mahoni",
    "database.password": "mahoni",
    "database dbname": "mahoni",
    "database.server.name": "postgres",
    "key.converter": "io.confluent.connect.avro.AvroConverter",
    "value.converter": "io.confluent.connect.avro.AvroConverter",
    "publication.autocreate.mode": "all_tables",
    "key.converter.schema.registry.url": "http://34.128.127.171:8081",
    "value.converter.schema.registry.url": "http://34.128.127.171:8081",
    "transform": "unwrap",
    "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState"
  }
}
```

Kode 5.4 Contoh Konfigurasi Debezium *Source Connector*

²⁵ <https://github.com/Mahoni-Smart-City/mahoni-core/tree/main/debezium/connectors>

<input type="checkbox"/>	Topic Name	Partitions	Out of sync replicas	Replication Factor	Number of messages	Size	<input type="button" value="..."/>
<input type="checkbox"/>	postgres.public.air_sensors	1	0	1	21	1 KB	<input type="button" value="..."/>
<input type="checkbox"/>	postgres.public.locations	1	0	1	267	23 KB	<input type="button" value="..."/>
<input type="checkbox"/>	postgres.public.merchants	1	0	1	434	143 KB	<input type="button" value="..."/>
<input type="checkbox"/>	postgres.public.or_generators	1	0	1	28	4 KB	<input type="button" value="..."/>
<input type="checkbox"/>	postgres.public.redeem_vouchers	1	0	1	282	62 KB	<input type="button" value="..."/>
<input type="checkbox"/>	postgres.public.trips	1	0	1	818	204 KB	<input type="button" value="..."/>
<input type="checkbox"/>	postgres.public.users	1	0	1	368	104 KB	<input type="button" value="..."/>
<input type="checkbox"/>	postgres.public.vouchers	1	0	1	483	114 KB	<input type="button" value="..."/>

Gambar 5.7 Daftar *Topic* yang Dibuat oleh *Connector Debezium*

Seluruh *connector* berhasil dibuat jika sudah terbuat *topic* untuk masing-masing tabel. Secara *default*, Debezium akan memberi nama *topic* <HOST>. <DBNAME>. <TABLE_NAME>. Daftar *topic* yang telah dibuat dari seluruh tabel dapat dilihat pada Gambar 5.7.

5.3.1.2.3 Kafka Connect dan Sink Connector

Di sisi lain CDC, perlu dilakukan konfigurasi Kafka Connect dan *sink connector* untuk menghubungkan Kafka dengan Cassandra dan *data warehouse*. Tak berbeda jauh dengan Debezium, Kafka Connect juga dapat dikonfigurasi menggunakan Docker *container*. Sebelum membuat Kafka Connect, harus diunduh terlebih dahulu *plugin* yang diperlukan. Untuk Cassandra, penulis menggunakan *plugin* yang dikembangkan oleh Lenses.io dan dapat diunduh dari *repository*²⁶ mereka, sedangkan *plugin* untuk Google Big Query dikembangkan oleh WePay dan dapat diunduh dari Confluent Hub²⁷. Kode 5.5 berisi konfigurasi docker-compose untuk membuat *container* Kafka Connect. Untuk memindahkan *plugin-plugin* yang telah diunduh oleh *host* ke Docker *container* perlu diatur *volumes* dengan menambahkan <LOKASI_PLUGIN_PADA_HOST>:<LOKASI_PADA_CONTAINER>. Setelah itu perlu juga diatur variabel **CONNECT_PLUGIN_PATH** yang mengarah pada folder yang berisi *plugins*. Konfigurasi lainnya tidak jauh berbeda dengan konfigurasi pada

²⁶ <https://github.com/lensesio/stream-reactor/releases/tag/4.2.0>

²⁷ https://www.confluent.io/hub/wepay/kafka-connect-bigquery?_gl=1*1mhpk2z*_ga*MTg2NzkyNTk2Mi4xNjc5NTU4NzE2*_ga_D2D3EGKSGD*MTY4NDk4NTUyOC4yMC4wLjE2ODQ5ODU1MjguNjAuMC4w&_ga=2.219046238.612297627.1684835311-1867925962.1679558716

Debezium. Kafka Connect diatur untuk berjalan pada *port* 8888 agar tidak terjadi bentrok dengan Debezium.

```
services:
  kafka-connect:
    image: confluentinc/cp-kafka-connect
    environment:
      CONNECT_BOOTSTRAP_SERVERS: PLAINTEXT://kafka:29092,PLAINTEXT://kafka-2:29092,PLAINTEXT://kafka-3:29092
      CONNECT_GROUP_ID: kafka-connect-group-id
      CONNECT_REST_ADVERTISED_HOST_NAME: kafka-connect
      CONNECT_REST_ADVERTISED_PORT: 8084
      CONNECT_CONFIG_STORAGE_TOPIC: kafka-connect-configs
      CONNECT_OFFSET_STORAGE_TOPIC: kafka-connect-offsets
      CONNECT_STATUS_STORAGE_TOPIC: kafka-connect-status
      CONNECT_KEY_CONVERTER: io.confluent.connect.avro.AvroConverter
      CONNECT_VALUE_CONVERTER: io.confluent.connect.avro.AvroConverter
      CONNECT_KEY_CONVERTER_SCHEMA_REGISTRY_URL: http://kafka-schema-registry:8081
      CONNECT_VALUE_CONVERTER_SCHEMA_REGISTRY_URL: http://kafka-schema-registry:8081
    CONNECT_PLUGIN_PATH: '/data/connectors'
    ports:
      - "9090:9090"
      - "8888:8083"
    volumes:
      - ./cassandra-sink/kafka-connect/plugins:/data/connectors
```

Kode 5.5 Konfigurasi *docker-compose* Kafka Connect

Setelah Kafka Connect berhasil berjalan dan memiliki *plugins* yang diperlukan, dapat dimulai proses pembuatan *connector*. Proses pembuatan *connector* mirip dengan yang ada di subbab Debezium, yaitu dengan mengirim POST *request* ke *endpoint* <HOST>:8888/connectors.

```
{
  "name": "user-sink",
  "config": {
    "connector.class": "com.datamountaineer.streamreactor.connect.cassandra.sink.CassandraSinkConnector",
    "topics": "postgres.public.users",
    "connect.cassandra.host": "cassandra-db",
    "connect.cassandra.port": "9042",
    "connect.cassandra.key.space": "mahoni",
    "connect.cassandra.table": "users",
    "connect.cassandra.contact.points": "cassandra-db",
    "connect.cassandra.kcql": "INSERT INTO users SELECT id, sex, year_of_birth FROM postgres.public.users"
  }
}
```

Kode 5.6 Contoh Konfigurasi Cassandra Sink Connector

Konfigurasi untuk Cassandra diadaptasi dari dokumentasi Aiven²⁸ dan contoh konfigurasi dapat dilihat pada Kode 5.6. Pada setiap konfigurasi perlu ditulis *topic* apa saja yang akan dikonsumsi. Kemudian variabel **connect.cassandra.kcql** berisi perintah *query* yang dilakukan untuk memasukkan data dari *events* ke tabel pada Cassandra. Seluruh konfigurasi tabel lain dapat dilihat di *repository*²⁹.

```
{
  "name": "user-cdc-kcbq",
  "config": {
    "connector.class": "com.wepay.kafka.connect.bigquery.BigQuerySinkConnector",
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "tasks.max" : "1",
    "topics" : "postgres.public.users",
    "sanitizeTopics" : "true",
    "autoCreateTables" : "true",
    "allowNewBigQueryFields" : "true",
    "allowBigQueryRequiredFieldRelaxation" : "true",
    "schemaRetriever" :
      "com.wepay.kafka.connect.bigquery.retrieve.IdentitySchemaRetriever",
      "project" : "mahoni-387706",
      "defaultDataset" : "user",
      "keyfile" : "/home/appuser/key/mahoni-387706-969252dea32e.json"
  }
}
```

Kode 5.7 Contoh Konfigurasi Google Big Query Sink Connector

Di sisi lain, konfigurasi Google Big Query dapat dilihat pada Kode 5.7. Pada konfigurasi ini Google Big Query akan membuat tabel baru jika diperlukan untuk mempermudah proses pengembangan.

5.3.2 Servis Web

Setiap servis aplikasi Mahoni dikembangkan dengan *framework* Spring Boot dan *deploy* pada Google Cloud Platform *compute engine* bertipe *e2-small* dengan spesifikasi 2 vCPU dan 2 GB *memory*. Spring Kafka adalah *library* utama yang digunakan untuk menghubungkan servis dengan *message broker*. KafkaStreams juga digunakan untuk implementasi KTable pada beberapa servis tertentu. Terdapat pula *library* pembantu lain seperti Avro dan *spring data*.

²⁸ <https://docs.aiven.io/docs/products/kafka/kafka-connect/howto/cassandra-streamreactor-sink>

²⁹ <https://github.com/Mahoni-Smart-City/mahoni-core/tree/main/cassandra-sink/kafka-connect/config>

5.3.2.1 Kafka Producer dan Consumer

Library Spring Kafka memberikan fleksibilitas yang tinggi dalam membantu konfigurasi Kafka *clients*. Pada penelitian ini penulis menggunakan konfigurasi yang diletakkan pada *application.yml*. Spring Kafka akan secara otomatis melakukan konfigurasi *producer* dan *consumer* berdasarkan *file* tersebut dan dapat langsung menggunakannya.

```
spring:
  kafka:
    bootstrap.servers:
      34.128.127.171:9092,34.128.127.171:9093,34.128.127.171:9094
      schema.registry.url: http://34.128.127.171:8081
    producer:
      bootstrap-servers: ${spring.kafka.bootstrap.servers}
      key-serializer:
        org.apache.kafka.common.serialization.StringSerializer
        value-serializer: io.confluent.kafka.serializers.KafkaAvroSerializer
        properties:
          schema.registry.url: ${spring.kafka.schema.registry.url}
    consumer:
      bootstrap-servers: ${spring.kafka.bootstrap.servers}
      key-deserializer:
        org.apache.kafka.common.serialization.StringDeserializer
        value-deserializer:
          io.confluent.kafka.serializers.KafkaAvroDeserializer
          properties:
            schema.registry.url: ${spring.kafka.schema.registry.url}
            specific.avro.reader: true
```

Kode 5.8 Konfigurasi *application.yml* pada Servis *User*

Kode 5.8 memperlihatkan potongan salah satu contoh konfigurasi pada servis *user*. Kode lengkap dapat dilihat di *repository* pada *file* *application.yml* untuk masing-masing servis. Pada konfigurasi tersebut nilai **schema.registry.url** mengarah pada *schema registry* Mahoni dan **specific.avro.reader** diset *true* untuk secara otomatis melakukan deserialisasi dengan *schema* yang telah terdaftarkan dan diperlukan juga class **KafkaAvroSerializer** karena *schema* yang dipakai memiliki format Avro. Di konfigurasi juga dituliskan ketiga Kafka *broker* yang ada, hal dilakukan agar servis tetap bisa terhubung ke Kafka jika satu *broker* mati.

```

@.Autowired
KafkaTemplate<String, UserPointSchema> kafkaTemplate;

public void send(User user, Integer point, String lastModifiedBy) {
    String id = UUID.randomUUID().toString();

    UserPointSchema event = UserPointSchema.newBuilder()
        .setEventId(id)
        .setTimestamp(parseTimestamp(LocalDateTime.now()))
        .setUserId(user.getId().toString())
        .setPrevPoint(user.getPoint() - point)
        .setPoint(user.getPoint())
        .setLastModifiedBy(lastModifiedBy)
        .build();

    log.info("Sending event to " + KafkaTopic.USER_POINT_TOPIC + " with
payload: " + event.toString());
    kafkaTemplate.send(new ProducerRecord<>("user-point-topic",
partition(user.getId()), id, event));
}

```

Kode 5.9 Pengiriman Event Menggunakan KafkaTemplate

Untuk menggunakan Kafka *producer*, penulis dapat langsung menggunakan *class KafkaTemplate* yang telah disediakan. **KafkaTemplate** mengandung tipe data dari *key* dan *value* pada *event* yang akan dikirim. Pengiriman *event* dapat dilakukan dengan memanggil fungsi **send()** seperti pada contoh kode 5.9.

```

private int partition(String identifier) {
    return Utils.toPositive(Utils.murmur2(identifier.getBytes())) %
USER_POINT_TOPIC_PARTITION;
}

```

Kode 5.10 Perhitungan Partisi Manual Berdasarkan *userId*

Seperti yang dijelaskan pada subbab rancangan Kafka *topic*, perlu dilakukan penentuan partisi secara manual agar data historis tetap terurut di dalam partisi yang sama. Implementasi tersebut dapat dilihat pada Kode 5.10, di mana digunakan *library* Murmur untuk menghitung *hash* dari *identifier*. Nilai *identifier* yang diberikan adalah *id* dari *user* terkait.

```

@KafkaListener( topics = "user-point-topic", groupId = "trip-service-group-id", containerFactory = "kafkaListenerContainerFactory")
public void consumePoint(ConsumerRecord<String, UserPointSchema> record) {
    log.info("Received event: " + record.value());
    UserPointSchema userPoint = record.value();
    // Run business process
    ...
}

```

Kode 5.11 Contoh Consumer yang *Subscribe* ke user-point-topic

Pembuatan sebuah *consumer* dapat menggunakan anotasi **@KafkaListener**. *Group id* perlu ditetapkan untuk setiap *consumer* untuk mengatur pembagian partisi. Pada penelitian ini setiap *listener* memiliki *group id* sesuai dengan nama servis masing-masing. Kode 5.11 memperlihatkan contoh pembuatan *consumer* pada servis *trip* yang *subscribe* ke *user-point-topic*.

5.3.2.2 KTable

KTable API hanya tersedia pada *library* KafkaStreams, maka dari itu perlu dilakukan konfigurasi lagi untuk menghubungkannya dengan Kafka *cluster*. Pada bagian ini penulis menjelaskan cara membuat KTable pada servis *trip* untuk mengonsumsi *air-quality-compacted-topic* dan mengubahnya menjadi *state store* agar bisa diakses menggunakan *key*. Terdapat tiga tahap yang perlu dilakukan yaitu pembuatan KafkaStreamsConfiguration, pembuatan topologi, dan pengambilan data dari *state store*. Kode utuh dapat dilihat pada Lampiran 14 atau *repository*³⁰.

```

@EnableKafka
@EnableKafkaStreams
public class TripServiceStream {
    ...
    @Bean(name =
KafkaStreamsDefaultConfiguration.DEFAULT_STREAMS_CONFIG_BEAN_NAME)
    public KafkaStreamsConfiguration kafkaStreamsConfiguration() {
        Map<String, Object> props = new HashMap<>();
        props.put(APPLICATION_ID_CONFIG, "trip-streams");
        props.put(GROUP_ID_CONFIG, "trip-streams-group-id");
        props.put(BOOTSTRAP_SERVERS_CONFIG, bootstrapAddress);

```

Kode 5.12 Potongan Kode Konfigurasi KafkaStreams.

³⁰ <https://github.com/Mahoni-Smart-City/mahoni-core/blob/main/trip-service/src/main/java/com/mahoni/tripservice/trip/config/KafkaConfiguration.java>

```

    props.put(DEFAULT_KEY_SERDE_CLASS_CONFIG,
Sedes.String().getClass().getName());
    props.put(DEFAULT_VALUE_SERDE_CLASS_CONFIG, SpecificAvroSerde.class);
    props.put(SCHEMA_REGISTRY_URL_CONFIG, schemaRegistryUrl);
    props.put(STATE_DIR_CONFIG, "./data/trip-service/store");
    return new KafkaStreamsConfiguration(props);
}

...

```

Kode 5.12 Potongan Kode Konfigurasi KafkaStreams. (lanjutan)

Pada tahap pertama ini penulis menggunakan konfigurasi menggunakan kode untuk memperlihatkan cara lain dalam menghubungkan servis ke Kafka. Seluruh konfigurasi diletakkan pada sebuah **HashMap** yang berisi *key* dan *value*. Diperlukan juga anotasi **@EnableKafka** dan **@EnableKafkaStreams** untuk memberi tahu Spring Boot bahwa akan digunakan *library* KafkaStreams seperti pada potongan Kode 5.12.

```

...
@Autowired
@Bean
public KTable<String, AirQualityProcessedSchema>
buildPipelineAirQuality(StreamsBuilder streamsBuilder) {
    Map<String, Object> props = new HashMap<>();
    props.put(SCHEMA_REGISTRY_URL_CONFIG, schemaRegistryUrl);
    Serde<String> stringSerde = Serdes.String();
    stringSerde.configure(props, true);
    SpecificAvroSerde<AirQualityProcessedSchema> avroSerde = new
SpecificAvroSerde<>();
    avroSerde.configure(props, false);

    return streamsBuilder
        .table(AIR_QUALITY_COMPACTED_TOPIC, Consumed.with(stringSerde,
avroSerde), Materialized.as("air-quality-state-store"));
}
...

```

Kode 5.13 Potongan Kode Pembuatan Topologi KTable.

Selanjutnya, tahap pembuatan topologi diperlukan untuk mengubah *air-quiality-compacted-topic* menjadi KTable. Hal yang perlu diperhatikan di sini adalah diperlukan fungsi **Materialized.as()** untuk menyimpan data *compacted topic* pada sebuah *state store*. *State store* ini membuat data di dalam *topic* dapat diakses menggunakan *key* dari *event* tersebut. Pada potongan Kode 5.13 terlihat bahwa KTable akan disimpan pada *state store* bernama *air-quailty-state-store*.

```

...
public AirQualityProcessedSchema getAirQuality(String id) {
    log.info("GET AIR QUALITY" + id);
    KafkaStreams kafkaStreams = factoryBean.getKafkaStreams();
    assert kafkaStreams != null;
    ReadOnlyKeyValueStore<String, AirQualityProcessedSchema> amounts =
kafkaStreams
        .store(StoreQueryParameters.fromNameAndType("air-quality-state-store",
QueryableStoreTypes.keyValueStore()));
    return amounts.get(id);}
}
...

```

Kode 5.14 Potongan Kode Pengambilan Data dari *State Store*.

Tahap terakhir dilakukan untuk mengambil data dari *state store*. Nama *state store* yang digunakan sesuai dengan pada topologi yang telah dibuat. Pada Kode 5.14 terlihat bahwa data dapat diakses menggunakan *state store* yang bernama **air-quality-state-store**.

5.3.2.3 Deployment

Proses *deployment* dibagi menjadi dua tahap yaitu pembuatan Docker *image* dan pembuatan *instance compute engine* di Google Cloud Platform. Servis yang telah selesai dikembangkan diubah terlebih dahulu menjadi Docker *image*. Untuk melakukan hal tersebut perlu di-*compile* terlebih dahulu proyek Spring Boot menjadi *file JAR*. Kemudian diperlukan juga sebuah Dockerfile untuk membuat Docker *image* dari *file JAR*. Setelah *image* berhasil terbuat, maka perlu diunggah ke *repository* yang berada pada Docker hub. Seluruh Docker *image docker* aplikasi Mahoni dapat dilihat pada Docker hub atas nama zsaschz³¹. Kode 5.15 menunjukkan Dockerfile untuk membuat Docker *image*, sedangkan Kode 5.16 berisi contoh *script* untuk membuat dan mengunggah Docker *image*.

```

FROM openjdk:17-alpine
RUN apk upgrade --no-cache && apk add --no-cache libstdc++
CMD ["java", "-jar", "/usr/share/mahoni/mahoni.jar"]
ARG JAR_FILE # Add the service itself
COPY target/${JAR_FILE} /usr/share/mahoni/mahoni.jar

```

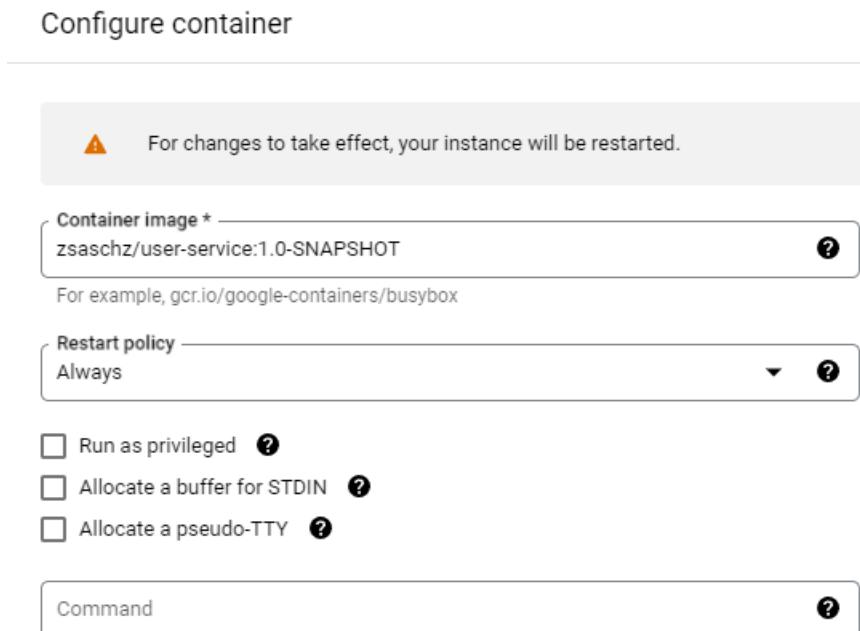
Kode 5.15 Dockerfile untuk Membuat Docker *image*

³¹ <https://hub.docker.com/repositories/zsaschz>

```
maven clean package
docker build zsaschz/user-service:1.0-SNAPSHOT
docker push zsaschz/user-service:1.0-SNAPSHOT
```

Kode 5.16 Script untuk Membuat dan Mengunggah Docker *image*

Tahap selanjutnya adalah pembuatan *instance* di Google Cloud Platform. Proses ini cukup mudah karena dapat memanfaatkan konfigurasi *container* langsung saat membuat *instance*. Fitur ini dapat digunakan dengan menekan tombol “Deploy Container” dan memberikan nama *image* servis pada bagian “Container Image” seperti pada Gambar 5.8.



Gambar 5.8 Konfigurasi *Container* pada Google Cloud Platform *Compute Engine*

Hal terakhir yang perlu dilakukan adalah mengatur *firewall* pada setiap servis. Namun, untuk mempermudah dan mempercepat proses pengembangan aplikasi penulis memilih untuk membuka akses seluruh *firewall*.

5.4 Evaluasi Arsitektur *Event-Driven*

Sesuai dengan yang dijelaskan pada subbab skenario pengujian arsitektur *event-driven*. Dilakukan evaluasi performa yang dinilai secara kuantitatif dengan melihat matriks *response time*, *response* per detik, maksimal *response time*, dan *events* per detik. Setelah

itu dilakukan pula evaluasi *loosely-coupled* dengan demonstrasi integrasi servis baru pada aplikasi Mahoni.

5.4.1 Evaluasi Performa

Penulis menggunakan alat bernama Locust yang dapat digunakan untuk menyimulasikan beberapa *user* melakukan *request* sekaligus. Locust secara otomatis menghitung *request* per detik dan *response time* untuk setiap *endpoint* sehingga sangat mempermudah proses evaluasi. Tabel 5.5 menunjukkan rangkuman hasil evaluasi performa masing-masing skenario. Gambar 5.9—5.11 memperlihatkan detail masing-masing skenario dan hasil lengkap tes dapat diakses pada *repository mahoni-core*³².

Tabel 5.5 Hasil Evaluasi Performa

Skenario	Total Events	Events per Detik	Total Request	Response per Detik	Rata-rata Response Time (ms)	Maksimal response time (ms)
1	52545	437,875	19884	165.8	1056	11907
2	185269	1543,908333	21353	177.9	945	23262
3	1385328	11544,4	21536	179.5	939	29720

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/air-quality/loc/[location]	892	0	30	24	120	16	7.4	0.0
GET	/air-sensors/[sensor_id]	892	0	41	22	273	203	7.4	0.0
POST	/api/v1/auth/register	849	0	2166	121	10232	208	7.1	0.0
GET	/api/v1/qr-generators/[qrgenerator_id]/generate-qrcode	4517	0	1444	29	11405	934	37.7	0.0
GET	/api/v1/qr-generators/decode-qrcode	4516	0	196	27	2001	152	37.7	0.0
POST	/api/v1/redeem	662	0	1003	29	4337	257	5.5	0.0
POST	/api/v1/redeem/[redeem_voucher_id]	656	0	821	25	5550	295	5.5	0.0
POST	/api/v1/trips	4447	0	1544	28	11907	450	37.1	0.0
POST	/api/v1/users	812	6	51	12	311	196	6.8	0.1
POST	/api/v1/vouchers	829	0	2370	30	9677	297	6.9	0.0
POST	/api/v1/vouchers/detail	812	0	1980	31	8926	827	6.8	0.0
Aggregated		19884	6	1056	12	11907	438	165.8	0.1

Gambar 5.9 Detail Hasil Evaluasi Performa Skenario Pertama

³² <https://github.com/Mahoni-Smart-City/mahoni-core/tree/main/load-test/results>

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/air-quality/loc/[location]	928	0	28	24	110	16	7.7	0.0
GET	/air-sensors/[sensor_id]	928	0	37	22	269	203	7.7	0.0
POST	/api/v1/auth/register	915	3	2828	24	17208	207	7.6	0.0
GET	/api/v1/qr-generators/[qrgenerator_id]/generate-qrcode	4863	1	881	13	6462	928	40.5	0.0
GET	/api/v1/qr-generators/decode-qrcode	4854	0	193	26	3007	152	40.4	0.0
POST	/api/v1/redeem	762	0	2001	28	11401	257	6.3	0.0
POST	/api/v1/redeem/[redeem_voucher_id]	736	0	1569	25	14989	295	6.1	0.0
POST	/api/v1/trips	4845	0	955	27	7392	450	40.4	0.0
POST	/api/v1/users	830	7	51	11	301	196	6.9	0.1
POST	/api/v1/vouchers	862	0	3187	29	23262	297	7.2	0.0
POST	/api/v1/vouchers/detail	830	0	2665	30	18508	827	6.9	0.0
Aggregated		21353	11	945	11	23262	437	177.9	0.1

Gambar 5.10 Detail Hasil Evaluasi Performa Skenario Kedua

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/air-quality/loc/[location]	988	0	29	23	89	16	8.2	0.0
GET	/air-sensors/[sensor_id]	988	0	43	22	280	203	8.2	0.0
POST	/api/v1/auth/register	942	3	3326	16	24340	207	7.8	0.0
GET	/api/v1/qr-generators/[qrgenerator_id]/generate-qrcode	4849	0	583	25	2438	928	40.4	0.0
GET	/api/v1/qr-generators/decode-qrcode	4849	0	130	24	1074	152	40.4	0.0
POST	/api/v1/redeem	777	0	2140	26	19405	257	6.5	0.0
POST	/api/v1/redeem/[redeem_voucher_id]	757	0	2116	25	21504	295	6.3	0.0
POST	/api/v1/trips	4847	0	619	27	3342	451	40.4	0.0
POST	/api/v1/users	820	9	70	11	378	196	6.8	0.1
POST	/api/v1/vouchers	899	0	4705	29	29004	297	7.5	0.0
POST	/api/v1/vouchers/detail	820	0	3683	29	29720	827	6.8	0.0
Aggregated		21536	12	939	11	29720	435	179.5	0.1

Gambar 5.11 Detail Hasil Evaluasi Performa Skenario Ketiga

Bila ditelusuri lebih lanjut, dapat dilihat bahwa terdapat perbedaan *response time* untuk seluruh POST request. Terdapat kenaikan *response time* sebesar 36% antara skenario satu dan dua. Kemudian, terdapat kenaikan lagi sebesar 26,5% antara skenario dua dan tiga. Hal yang menarik adalah analisis terhadap *memory* dan CPU masing-masing Kafka broker pada saat melakukan skenario tidak menunjukkan adanya *bottleneck*. Tabel 5.6—5.8 memperlihatkan bahwa tidak adanya perbedaan signifikan pada CPU *usage* dan

memory antar skenario. Kenaikan *memory* saat *idle* dan maksimum selalu berkisar pada sekitar 50MB pada seluruh skenario. CPU *usage* juga tidak pernah melebihi 30% meskipun pada skenario ketiga yang memiliki jumlah *events per second* 120 kali lebih besar dari skenario pertama.

Tabel 5.6 *Memory* dan *CPU Usage* pada Skenario Pertama

Broker No.	Memory on Idle	Maximum Memory	CPU Usage on Idle	Maximum CPU Usage
1	325MB	360MB	1,5%	15%
2	370MB	420MB	1,6%	25%
3	330MB	380MB	1,5%	13%

Tabel 5.7 *Memory* dan *CPU Usage* pada Skenario Kedua

Broker No.	Memory on Idle	Maximum Memory	CPU Usage on Idle	Maximum CPU Usage
1	330MB	390MB	1,6%	14%
2	390MB	443MB	1,6%	22%
3	353MB	402MB	1,5%	15%

Tabel 5.8 *Memory* dan *CPU Usage* pada Skenario Ketiga

Broker No.	Memory on Idle	Maximum Memory	CPU Usage on Idle	Maximum CPU Usage
1	366MB	398MB	1,6%	22%
2	400MB	480MB	2,4%	27%
3	372MB	422MB	2,1%	20%

Oleh karena itu, penulis menyimpulkan bahwa penurunan performa bisa jadi terjadi karena *bottleneck* pada *schema registry* atau Spring Boot itu sendiri. *Schema registry* dapat menjadi *bottleneck* karena hanya ada satu *node* untuk seluruh Kafka *cluster*. Di sisi lain, pada Spring Boot sendiri terdapat limitasi dari jumlah *thread* yang dapat dibuat untuk melayani *request*.

Meskipun demikian, hasil evaluasi juga menunjukkan bahwa *throughput* tinggi pada *message broker* tidak terlalu berpengaruh performa aplikasi secara keseluruhan. Rata-rata

response time pada masing-masing skenario tetap berkisar pada 900—1000 *milisecond* dan *request* per detik berkisar pada 160—180 pada seluruh skenario pengujian.

5.4.2 Evaluasi Sifat *Loosely-Coupled*

Servis loyalitas dibuat sama seperti servis lain pada aplikasi Mahoni yaitu menggunakan Spring Boot dan *library* Spring Kafka. Kode 5.17 menunjukkan konfigurasi yang digunakan untuk menghubungkan servis dengan Kafka. Nilai *auto-offset-reset* perlu diatur menjadi *earliest* untuk memastikan *consumer* membaca seluruh *event* dari awal. Informasi yang diperlukan untuk tahap ini adalah *url* dari *schema registry* dan minimal satu Kafka *broker*. Kafka memastikan bahwa jika *consumer* terhubung ke salah satu *broker*, maka *consumer* tersebut dapat menerima seluruh data di dalam Kafka *cluster*.

```
spring:
  kafka:
    bootstrap.servers: 34.128.127.171:9092
    schema.registry.url: http://34.128.127.171:8081
    producer:
      bootstrap-servers: ${spring.kafka.bootstrap.servers}
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer: io.confluent.kafka.serializers.KafkaAvroSerializer
      properties:
        schema.registry.url: ${spring.kafka.schema.registry.url}
    consumer:
      bootstrap-servers: ${spring.kafka.bootstrap.servers}
      key-deserializer:
        org.apache.kafka.common.serialization.StringDeserializer
      value-deserializer:
        io.confluent.kafka.serializers.KafkaAvroDeserializer
      auto-offset-reset: earliest
      properties:
        schema.registry.url: ${spring.kafka.schema.registry.url}
        specific.avro.reader: true
```

Kode 5.17 Konfigurasi Servis Loyalitas

Selanjutnya untuk mengetahui bentuk data yang diproses diperlukan informasi dari *topic* dan isi *event* di dalamnya, dan di sinilah peran penting dari *schema registry*. Dengan mengakses Kafka UI dan melihat *schema* dari *user-point-topic* dapat ditentukan proses apa saja yang diperlukan untuk melakukan perhitungan poin loyalitas. Kode 5.18 memperlihatkan *schema* pada *user-point-topic*. Pada tahap ini diperlukan informasi tentang *topic* dan *schema*-nya.

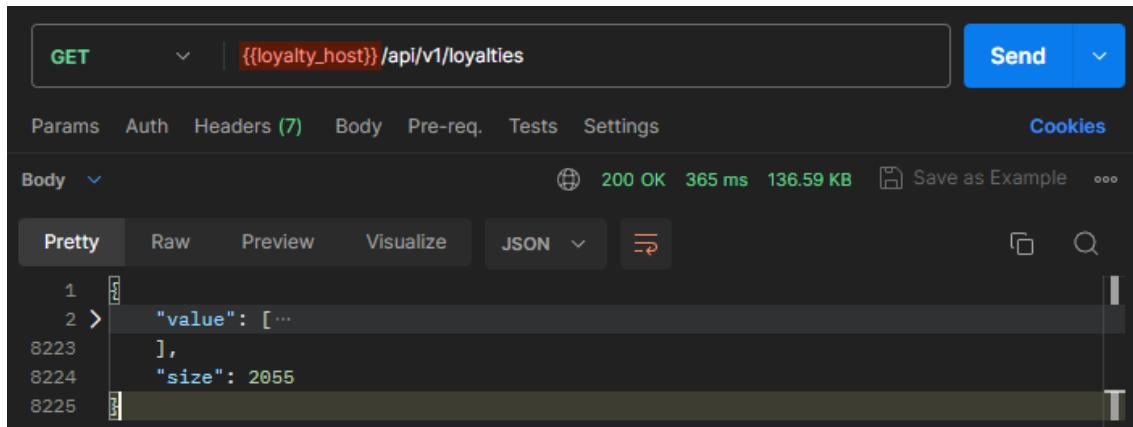
```
{
    "type": "record",
    "name": "UserPointSchema",
    "namespace": "com.mahoni.schema",
    "fields": [
        {
            "name": "eventId",
            "type": {
                "type": "string",
                "avro.java.string": "String"
            }
        },
        {
            "name": "timestamp",
            "type": "long"
        },
        {
            "name": "userId",
            "type": {
                "type": "string",
                "avro.java.string": "String"
            }
        },
        {
            "name": "prevPoint",
            "type": "int"
        },
        {
            "name": "point",
            "type": "int"
        },
        {
            "name": "lastModifiedBy",
            "type": {
                "type": "string",
                "avro.java.string": "String"
            }
        }
    ]
}
```

Kode 5.18 Schema pada *user-point-topic*

Servis di-deploy pada *instance* yang serupa dengan servis lain. Pada saat selesai proses *deployment*, dimulailah proses konsumsi seluruh *event* dari *user-point-topic*. Terdapat total 10005 total *event* pada *topic* dari 2055 *user* berbeda. Proses konsumsi memerlukan waktu 3,04 detik atau setara dengan sekitar 3291 *event* per detik. Bukti log lengkap servis dapat dilihat di *repository*³³. Setelah itu servis sudah mendapatkan data terbaru tentang

³³ <https://github.com/Mahoni-Smart-City/mahoni-core/blob/main/coupling-test/loyalty-service-log.txt>

loyalitas setiap *user* dari data historis yang disimpan pada *user-point-topic*. Gambar 5.15 memperlihatkan *response* servis loyalitas setelah mendapatkan seluruh data.



Gambar 5.15 *Response* Servis Loyalitas

Evaluasi ini mengungkapkan bahwa informasi minimal yang dibutuhkan untuk integrasi komponen baru pada aplikasi Mahoni adalah (1) *url* satu Kafka *broker*, (2) *url schema registry*, (3) *topic* yang ingin dikonsumsi, dan (4) *schema* pada *topic* yang dikonsumsi. Evaluasi ini juga membuktikan bahwa arsitektur *event-driven* mahoni *loosely-coupled* karena seluruh informasi bisa didapatkan tanpa berinteraksi langsung dengan servis *user* sebagai domain yang memiliki data poin setiap *user*.

5.5 Kesimpulan Arsitektur *Event-Driven*

Pada bab ini telah dijelaskan proses perancang dan implementasi arsitektur *event-driven* untuk aplikasi Mahoni. Dua hal yang menjadi fokus utama penulis saat membuat arsitektur adalah untuk membuat arsitektur yang memiliki *throughput* tinggi dan bersifat *loosely-coupled*. Berdasarkan hasil evaluasi performa pada bagian 5.4.1, dapat disimpulkan bahwa arsitektur yang dibuat bisa mengakomodasi *throughput* sampai 10000 *events* per detik pada *message broker* dengan konstan tetap bisa mempertahankan 160-180 *request* per detik walaupun ada kenaikan *response time*.

Selain itu, hasil evaluasi sifat *loosely-coupled* arsitektur pada bagian 5.4.2 memberikan informasi minimum yang diperlukan untuk melakukan integrasi dengan sistem Mahoni. Seluruh informasi yang diperlukan bisa diperoleh tanpa berinteraksi langsung dengan domain pemilik data, hal ini membuat arsitektur yang dibuat bersifat *loosely-coupled*.

Schema registry dan *Kafka* merupakan dua komponen utama yang membuat hal ini dapat tercapai. Dengan demikian, integrasi komponen baru pada sistem Mahoni dapat dengan mudah dilakukan, apalagi melihat bahwa Mahoni adalah sebuah aplikasi kota cerdas yang terdiri dari banyak komponen berbeda.

Terlebih lagi, arsitektur yang dibuat juga telah mengintegrasikan komponen *big data* yang dengan menggunakan proses *change data capture*. Proses CDC menghasilkan data *stream* konstan terhadap perubahan *database*. Data *stream* tersebut kemudian dapat diolah dan diproses oleh komponen *stream processing* untuk banyak kebutuhan lain seperti *monitoring* atau *real-time analytics*.

Seluruh hal itu dicapai dengan tetap memperhatikan proses bisnis yang diperlukan aplikasi Mahoni. Arsitektur dibuat sedemikian sehingga tetap bisa menjaga konsistensi data antar servis dengan membuat alur khusus untuk mencapai konvergensi data seperti yang dijelaskan pada bagian 5.2.2.5. Meskipun demikian, penulis juga menemukan adanya *bottleneck* dan diberikan saran untuk penelitian atau implementasi selanjutnya agar tidak terjadi hal yang serupa pada subbab Saran.

BAB 6

ARSITEKTUR *BIG DATA* MAHONI

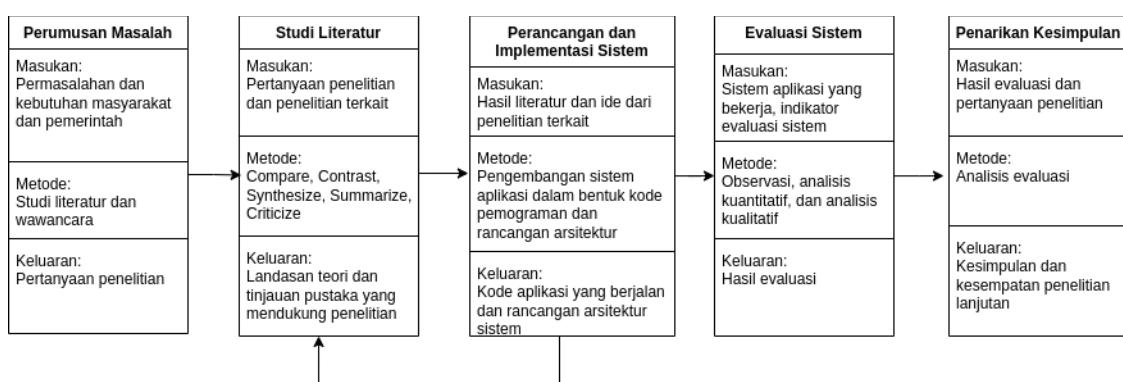
Arsitektur *big data* memiliki metodologi tersendiri dalam melakukan tahapan penelitian, skenario, dan matriks evaluasi. Arsitektur *big data* yang dibuat pada penelitian ini dibagi menjadi tiga tahap dalam proses pengembangannya yaitu praperancangan, perancangan, dan implementasi. Setelah selesai diimplementasi, arsitektur ini dievaluasi yang hasilnya dapat digunakan untuk pengembangan aplikasi ke depannya. Semua proses dalam pembuatan arsitektur *big data* dijelaskan pada bab ini.

6.1 Metodologi Penelitian

Penelitian untuk menyelesaikan masalah arsitektur *big data* ini menggunakan jenis penelitian eksperimental dengan pengujian berupa kuantitatif dan kualitatif. Penelitian ini menggunakan beberapa tahapan penelitian, skenario, dan matriks evaluasi yang dijelaskan pada subbab di bawah ini.

6.1.1 Tahapan Penelitian

Tahapan penelitian yang dilakukan pada penelitian ini memiliki tahapan yang sama seperti yang dijelaskan pada bab 3.2.1. Hal yang membedakan yaitu tahapan perumusan masalah. Metode yang dilakukan saat perumusan masalah adalah wawancara dan studi literatur.



Gambar 6.1 Tahapan Penelitian Arsitektur Big Data

Wawancara dilakukan dengan dinas-dinas terkait dengan permasalahan pada sistem aplikasi Mahoni yang berada di DKI Jakarta. Hasil dari wawancara tersebut dijelaskan pada bab 6.2. Selain itu, dilakukan studi literatur terkait dengan arsitektur *big data* dan

framework yang digunakan di dalam arsitektur tersebut. Proses pemilihan bentuk arsitektur dan *framework* yang digunakan dijelaskan pada bab 6.3. Keluaran yang dihasilkan dari tahapan ini adalah pertanyaan penelitian yang menjadi masukan pada tahapan selanjutnya.

6.1.2 Skenario Pengujian

Dilakukan beberapa pengujian untuk menguji arsitektur *big data* pada penelitian ini. Pengujian yang dilakukan dibagi menjadi dua yaitu pengujian sistem dan pengujian *dashboard*. Pengujian sistem dilakukan dengan uji fungsionalitas dan uji *scalability* dan *reliability*.

Pengujian fungsionalitas arsitektur dilakukan dengan melakukan *integration test*. *Integration test* merupakan pengujian untuk menguji koneksi antar komponen untuk memastikan bahwa komponen-komponen dapat terkoneksi dengan baik sesuai dengan hasil yang diinginkan. Selain itu pengujian *dashboard* bertujuan untuk menyesuaikan kebutuhan dari dinas-dinas DKI Jakarta yang dibuatkan beberapa *user story* dengan data-data yang ditampilkan oleh *dashboard*. Pembuatan *user story* dijelaskan pada bab 6.2.2.

Tabel 6.1 Skenario Pengujian *Scalability* dan *Reliability*

Nomor Skenario	JobManager	TaskManager	Data (data/detik)
1	1	3	13
2	1	3	39
3	1	3	65
4	1	3	130
5	1	5	13
6	1	5	39
7	1	5	65
8	1	5	130

Pengujian *scalability* dan *reliability* dilakukan untuk mengevaluasi kemampuan arsitektur dalam mengolah data dengan jumlah tertentu. Di dalam pengujian ini, arsitektur diuji dengan beberapa skenario pertumbuhan data dari data generator *Air Sensor* dan pertambahan beberapa *node* pada Flink dalam arsitektur *big data*. Asumsi yang digunakan pada pengujian ini adalah kemampuan pengolahan data sensor udara

menggambarkan kebutuhan minimum yang diterapkan untuk mengolah semua data pada sistem aplikasi Mahoni. Hal ini disebabkan pengolahan data sensor udara lebih kompleks dibandingkan dengan pengolahan data lain. Selain itu hanya data tersebut yang dapat diatur konsistensi jumlah data dibandingkan dengan data lainnya. Data perjalanan dan penggunaan *voucher* tidak konsisten jika dijadikan pengukuran dalam pengujian ini karena kemungkinan *delay* dari servis yang masuk ke dalam arsitektur *big data*. Pengujian dilakukan dengan skenario sesuai dengan Tabel 6.1.

6.1.3 Matriks Evaluasi

Skenario pengujian yang direncanakan memiliki beberapa matriks evaluasi sebagai pengukuran keberhasilan arsitektur yang sudah dibuat. Pengujian fungsionalitas arsitektur memiliki matriks yaitu keberhasilan koneksi antar *layer* arsitektur. Hal ini dapat dibuktikan dengan membuat *integration test* yang berhasil dan konektivitas yang ditunjukkan dengan masuknya data pada *framework* yang digunakan. Lalu untuk pengujian *dashboard* dilakukan pengecekan kesesuaian antara kebutuhan *user story* yang sudah dibuat dengan *dashboard*.

Pengujian *scalability* dan *reliability* menggunakan beberapa matriks evaluasi yang menjadi gambaran performa pada Flink yaitu perbandingan jumlah data antara data yang masuk pada pemrosesan pertama dan data yang keluar pada pemrosesan terakhir, total waktu ketika pemrosesan data mengalami kesibukan, penggunaan tertinggi CPU, dan penggunaan tertinggi memori. Perbandingan jumlah data yang diproses pada pemrosesan pertama dan terakhir dapat menggambarkan masalah yang terjadi pada pemrosesan data Flink. Masalah yang terjadi pada pemrosesan data juga tergambar pada waktu sibuk yang terjadi, penggunaan CPU, dan penggunaan memori.

6.2 Praperancangan Arsitektur Big Data

Arsitektur *big data* memiliki tujuan utama yaitu menyediakan data penggunaan aplikasi dan kualitas udara secara *real-time* berupa *dashboard* dan data laporan kepada pemerintah. Oleh karena itu, sebelum arsitektur ini dirancang dan diimplementasi diperlukan validasi kebutuhan kepada pihak pemerintah mengenai sistem aplikasi Mahoni. Proses validasi kebutuhan ini dilakukan dengan melakukan wawancara kepada dinas-dinas di DKI Jakarta yang terkait dengan sistem aplikasi Mahoni.

6.2.1 Pengumpulan Data Kebutuhan Pemerintah

Pihak pemerintah yang terkait dengan sistem aplikasi Mahoni yaitu pihak yang mengurus lingkungan hidup, transportasi, UMKM dan pengembangan kota cerdas. Dengan mengambil latar di DKI Jakarta, penulis menghubungi dinas di DKI Jakarta untuk melakukan wawancara. Dinas-dinas tersebut adalah Dinas Lingkungan Hidup DKI Jakarta, Dinas Perhubungan DKI Jakarta, Dinas Pariwisata dan Ekonomi Kreatif DKI Jakarta, dan Jakarta Smart City.

Penulis hanya berkesempatan untuk mewawancarai pihak Jakarta Smart City dan Dinas Perhubungan DKI Jakarta saja. Pihak Dinas Lingkungan Hidup DKI Jakarta dan Dinas Pariwisata dan Ekonomi Kreatif DKI Jakarta tidak memberikan respons terkait wawancara tersebut. Hasil wawancara digunakan untuk mendapatkan kebutuhan-kebutuhan dari dinas, data-data tambahan, pendapat, dan masukan untuk sistem aplikasi Mahoni.

6.2.1.1 Hasil Wawancara dengan Jakarta Smart City

Wawancara yang dilakukan dengan Jakarta Smart City dilakukan untuk mendapatkan informasi terkait konsep pengembangan kota cerdas yang dilakukan di DKI Jakarta. Pengembangan kota cerdas yang dilakukan oleh Jakarta Smart City berpedoman pada Kajian Strategi Daerah dan arahan gubernur. Pengembangan yang dilakukan berdasarkan konsep kota cerdas yaitu *smart mobile* dan *smart living* yang bertujuan untuk merespons keluhan masyarakat secara cepat.

Jakarta Smart City mengembangkan dua aplikasi yaitu Jaklapor dan Jaki yang berfungsi untuk menampung keluhan masyarakat serta menyediakan informasi terbaru kepada masyarakat. Informasi-informasi yang disediakan merupakan informasi yang didapatkan dari dinas-dinas yang ada di DKI Jakarta. Walaupun dalam pengembangannya Jakarta Smart City berpedoman pada peraturan dan arahan gubernur, namun Jakarta Smart City melakukan evaluasi dengan melibatkan masyarakat. Pendekatan ini berbeda dengan pengembangan aplikasi pada umumnya yang melibatkan masyarakat atau penggunanya dari awal sampai akhir pembuatan aplikasi. Hal ini dikarenakan Jakarta Smart City terikat oleh pemerintahan DKI Jakarta.

6.2.1.2 Hasil Wawancara dengan Dinas Perhubungan DKI Jakarta

Penggunaan transportasi umum di DKI Jakarta selama 3 tahun terakhir mengalami peningkatan. Walaupun sempat menurun pada 2 tahun lalu akibat pembatasan terkait dengan pandemi virus Corona, namun saat ini penggunaan transportasi umum kembali normal. Penggunaan transportasi umum kembali ini didorong karena beberapa faktor yaitu adanya kebijakan pembatasan kendaraan di beberapa ruas jalan serta faktor kenyamanan dan pelayanan transportasi umum yang lebih baik.

Transportasi umum yang dibawahi oleh Dinas Perhubungan DKI Jakarta adalah MRT, LRT, Transjakarta, bus wisata, bus sekolah, dan angkot Jaklingko. Dalam operasionalnya, Dinas Perhubungan DKI Jakarta menyimpan dan mengolah data perjalanan yang dilakukan oleh masyarakat saat melakukan *tap in* dan *tap out* saat menaiki kendaraan. Namun data yang dikumpulkan kurang akurat karena *tap* yang dilakukan sering dimanipulasi dengan melakukan *tap in* dan *tap out* secara langsung. Data yang dikumpulkan yaitu pergerakan posisi dan perjalanan setiap orang namun tidak detail sampai ke data pribadi masyarakat karena masih terkendala teknis dan peraturan.

Dinas Perhubungan DKI Jakarta menyadari bahwa penggunaan transportasi pribadi di DKI Jakarta memiliki kaitannya dengan buruknya kualitas udara. Oleh karena itu, dalam beberapa hal Dinas Perhubungan DKI Jakarta bekerja sama dengan Dinas Lingkungan Hidup DKI Jakarta untuk mengatasi hal ini salah satunya dengan melakukan uji emisi kendaraan. Berbicara kaitan antara penggunaan transportasi umum dengan kualitas udara, pihak Dinas Perhubungan DKI Jakarta mengapresiasi dan memberikan masukan terkait pengembangan sistem aplikasi Mahoni yang memiliki tujuan untuk meningkatkan kualitas udara dengan menggunakan transportasi umum.

6.2.2 Pembuatan *User Story Dashboard*

Setelah proses wawancara selesai dilakukan, kebutuhan data dari aplikasi Mahoni untuk setiap dinas dirangkum menjadi sebuah *user story*. Di dalam *user story* ini diidentifikasi kebutuhan dinas untuk pembuatan *dashboard* sebagai hasil dari pembuatan arsitektur *big data*. Untuk kebutuhan *dashboard* mengenai kualitas udara dan penggunaan *voucher*, penulis menggunakan asumsi akibat wawancara yang tidak berhasil dilaksanakan. Berikut adalah *user story* yang sudah dirangkum,

1. Pemantauan kualitas udara
 - a. Dinas Lingkungan Hidup DKI Jakarta ingin mengetahui kondisi kualitas udara secara langsung berdasarkan tempat sensor udara.
 - b. Dinas Lingkungan Hidup DKI Jakarta ingin mengetahui gambaran umum (maksimum, minimum, dan rata-rata) kondisi kualitas udara secara langsung berdasarkan tempat sensor udara.
2. Pemantauan penggunaan transportasi umum
 - a. Dinas Perhubungan DKI Jakarta ingin mengetahui jumlah pengguna yang melakukan *scan in* dan *scan out* secara langsung pada pemberhentian transportasi umum yang ada.
 - b. Dinas Perhubungan DKI Jakarta ingin mengetahui jumlah pengguna secara langsung pada suatu pemberhentian transportasi umum berdasarkan jenis kelamin pengguna.
 - c. Dinas Perhubungan DKI Jakarta ingin mengetahui secara langsung total pengguna yang memulai perjalanan, menyelesaikan perjalanan, dan gagal pada aplikasi Mahoni berdasarkan pemberhentian transportasi umum.
 - d. Dinas Perhubungan DKI Jakarta ingin mendapatkan laporan bulanan dari semua pemantauan langsung yang dilakukan.
3. Pemantauan penggunaan *voucher* Mahoni
 - a. Dinas Pariwisata dan Ekonomi Kreatif DKI Jakarta ingin mengetahui secara langsung jumlah pengguna *voucher* pada tipe *voucher* tertentu berdasarkan jenis kelamin pengguna secara langsung.
 - b. Dinas Pariwisata dan Ekonomi Kreatif DKI Jakarta ingin mengetahui jenis *voucher* yang dibeli oleh pengguna selama satu minggu.
 - c. Dinas Pariwisata dan Ekonomi Kreatif DKI Jakarta ingin mengetahui total pembelian *voucher* dalam waktu satu minggu.

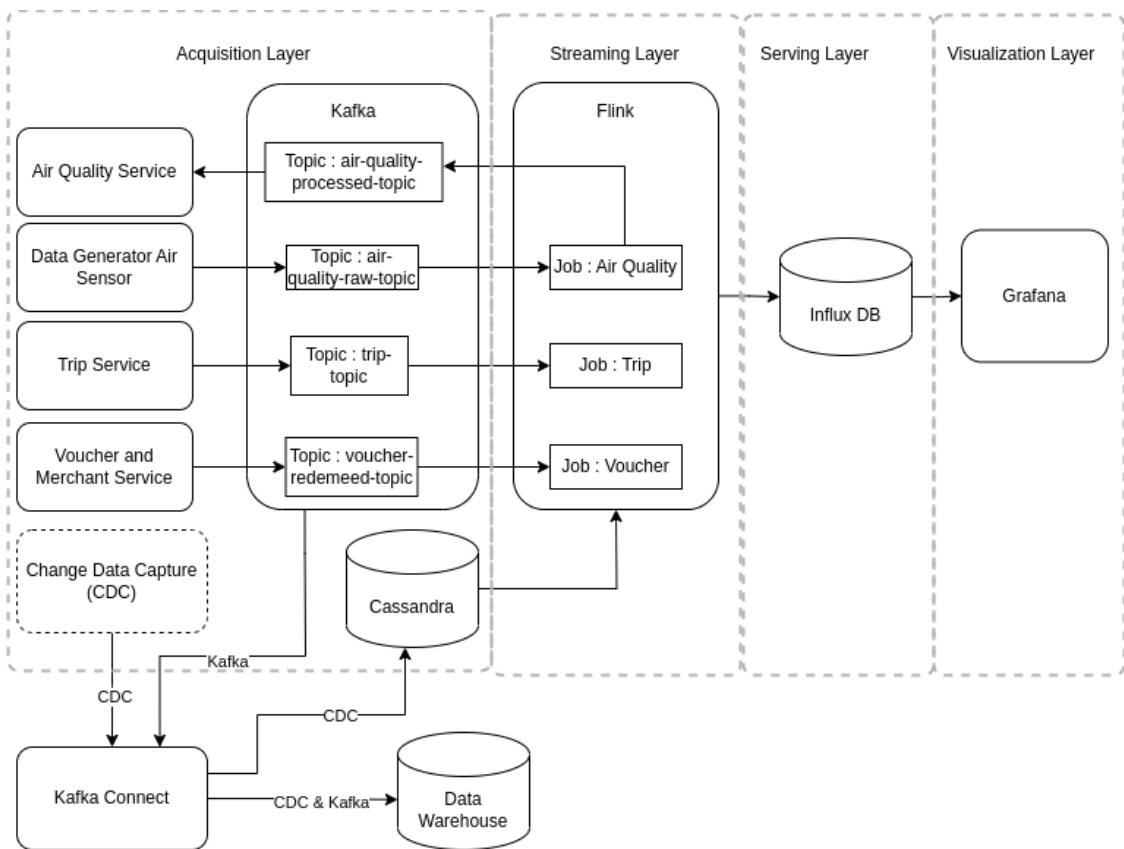
6.3 Perancangan Arsitektur *Big Data*

Perancangan arsitektur *big data* dilakukan setelah mengetahui kebutuhan pengguna melalui *user story* yang telah dibuat. Beberapa konsep dan *framework* yang digunakan pada perancangan arsitektur ini terinspirasi dari penelitian yang dikerjakan oleh Hazman F.K., (2020) namun dengan implementasi yang berbeda. Pada tahap ini dilakukan

pemilihan bentuk arsitektur, pemilihan *framework*, dan rancangan penyimpanan data arsitektur *big data*.

6.3.1 Rancangan Arsitektur *Big Data*

Arsitektur yang diterapkan pada penelitian ini menggunakan arsitektur Kappa. Pemilihan arsitektur ini berdasarkan kebutuhan sistem yang sesuai untuk melakukan pemrosesan data *stream*. Dengan kelebihan yang dimiliki, arsitektur Kappa dapat memproses data yang diterima secara langsung tanpa melakukan pencocokan data yang berpengaruh pada waktu pemrosesan data.



Gambar 6.2 Arsitektur *Big Data*

Gambar 6.2 menunjukkan arsitektur sistem aplikasi Mahoni yang dibangun dengan beberapa *layer* yang menerapkan arsitektur Kappa. *Streaming layer* dan *serving layer* memiliki sumber data yaitu *acquisition layer* dan diakhiri dengan *visualization layer* untuk menampilkan data melalui *dashboard*. Penjelasan lebih lanjut mengenai *layer-layer* yang ada di arsitektur ini dijelaskan pada subbab di bawah ini.

6.3.1.1 Acquisition Layer

Acquisition layer merupakan *layer* yang menjadi sumber data untuk diolah. Data-data yang disediakan oleh penelitian ini adalah data *stream* yang berasal dari data generator *Air Sensor*, aktivitas pengguna, dan data hasil *Change Data Capture* (CDC) yang disimpan di Cassandra. Data generator *Air Sensor* dan aktivitas pengguna diproduksi oleh Kafka dengan tiga topik yaitu *air-quality-raw-topic*, *trip-topic*, dan *voucher-redeemed-topic*. Masing-masing topik ini dikonsumsi oleh *streaming layer* melalui Flink dijelaskan pada subbab berikutnya.

Ketiga topik yang dikonsumsi dari Kafka merupakan data transaksional yaitu data yang berkaitan dengan aktivitas terkait dengan sistem. Data ini biasanya tidak memuat informasi yang detail, sehingga perlu dilengkapi dengan data lainnya. Data yang melengkapi data transaksional ini merupakan data non-transaksional yang berisi informasi yang sifatnya lebih statis. Data non-transaksional ini berasal dari *database* setiap servis yang disalin perubahannya menggunakan *Change Data Capture* (CDC). Hasil CDC ini disimpan di Cassandra yang menyediakan data untuk melengkapi data transaksional yang sedang diolah di *streaming layer*. Terdapat beberapa pilihan *database* yang dapat digunakan selain Cassandra. Untuk penjelasan lebih lanjutnya dijelaskan di bab 6.4.

6.3.1.2 Streaming Layer

Streaming layer berfungsi untuk melakukan pemrosesan data *stream* yang dikonsumsi dari topik Kafka dan dilengkapi dengan data dari Cassandra yang menyimpan hasil CDC. Pemrosesan data dalam penelitian ini menggunakan *framework* Flink. Flink memproses data dengan menjalankan *job* yang telah dibuat. *Job* merupakan serangkaian *task* atau tugas yang dilakukan selama pemrosesan data.

Terdapat tiga *job* yang dikerjakan oleh Flink sesuai dengan Gambar 6.2 yaitu *air quality job*, *trip job*, dan *voucher job*. Ketiga *job* ini melakukan pemrosesan data sesuai dengan kebutuhannya masing-masing. Pemrosesan data yang dilakukan oleh ketiga *job* ini adalah data *enrichment*. Data *enrichment* merupakan proses melengkapi data dengan *Stream-Table join* yang melengkapi data *stream* dengan data *table* yang berasal dari Cassandra. Seperti yang sudah dibahas sebelumnya, Cassandra menyimpan data non-transaksional

yang digunakan untuk melengkapi data transaksional dari Kafka. Proses ini dijelaskan lebih lanjut pada bab 6.4.2.

Selain melakukan data *enrichment*, setiap *job* juga melakukan *sink* data ke *serving layer*. Ketiga *job* ini disimpan data hasil pengolahannya di *serving layer* dan akan divisualisasi di *visualization layer*. Selain mengirimkan data ke *layer* tersebut, Flink juga melakukan *sink* ke Kafka untuk memproduksi hasil pemrosesan data kualitas udara. Hasil ini diproduksi dengan topik *air-quality-processed-topic* yang dikonsumsi oleh *Air Quality Service*.

6.3.1.3 Serve Layer dan Visualization Layer

Serve layer dalam arsitektur *big data* berfungsi sebagai tempat untuk menyimpan data yang sudah diproses pada *streaming layer* untuk diolah kembali. Di dalam penelitian ini, *serve layer* berfungsi untuk menyimpan data yang digunakan untuk visualisasi data berbentuk sebuah *dashboard*. *Dashboard* ini digunakan oleh pemerintah dalam memonitor kualitas udara, penggunaan transportasi dan penggunaan *voucher* di aplikasi Mahoni. *Framework* yang digunakan *serve layer* dalam penelitian ini adalah InfluxDB yang merupakan *time-series database*. Data disimpan di InfluxDB sesuai dengan rancangan data yang dijelaskan pada bab 6.3.3. Selain InfluxDB, terdapat beberapa *framework* lain berupa *time series database*. Untuk penjelasan pemilihan InfluxDB sebagai *database* dijelaskan pada bab 6.3.2.2.

Visualization layer berfungsi untuk menampilkan data yang telah disimpan di *serving layer* dalam bentuk sebuah *dashboard*. *Dashboard* yang diimplementasi dalam penelitian ini menggunakan Grafana. Grafana tersambung secara langsung dari InfluxDB untuk memberikan hasil *dashboard* secara *real-time*. *Dashboard* dibuat dari hasil *query* InfluxDB yang disesuaikan dengan *user story* yang telah dibuat pada bab 6.2.2. Pembuatan *dashboard* bertujuan agar pihak pemerintah dapat mengakses perubahan data secara *real-time*.

6.3.1.4 Data Warehouse

Data warehouse pada arsitektur *big data* berfungsi sebagai “*single version of truth*” sistem aplikasi Mahoni karena menyimpan semua data baik data transaksional dan non-

transaksional. Dilihat pada Gambar 6.2 *data warehouse* menyimpan data non-transaksional dari hasil CDC dan data transaksional dari Kafka. Kedua data ini dihubungkan melalui Kafka Connect sehingga semua aliran data terkumpul pada *data warehouse*. *Data warehouse* terpilih untuk menyimpan semua data dikarenakan memiliki ukuran penyimpanan yang besar dan kemampuan untuk melakukan *query* untuk mendapatkan data berbentuk sebuah laporan analisis.

Laporan analisis merupakan salah satu kebutuhan pemerintah dengan membuat laporan dengan rentang waktu satu bulan. Data dalam rentang waktu ini tersedia oleh *data warehouse* karena semua data tersimpan di sini dan tidak memiliki restriksi untuk melakukan penghapusan dalam jangka waktu tertentu. *Data warehouse* yang digunakan pada penelitian ini adalah BigQuery.

6.3.2 Pemilihan Framework

Terdapat beberapa pilihan *framework* yang dapat digunakan di dalam arsitektur *big data*. Namun pemilihan *framework* ini harus berdasarkan kebutuhan dan karakteristik sistem yang akan dibangun. Setiap *framework* memiliki karakteristik tersendiri sehingga memiliki kelebihan dan kekurangannya masing-masing dalam situasi dan kebutuhan yang berbeda. Maka dari itu, dilakukan pemilihan *framework* pada penelitian ini berdasarkan penelitian terdahulu untuk mendapatkan *framework* yang sesuai dengan kebutuhan.

6.3.2.1 Apache Flink

Streaming layer memiliki beberapa pilihan *framework* dengan pendekatan pemrosesan data yang berbeda-beda. Pendekatan yang berbeda ini membuat pemrosesan data *stream* secara *real-time* memiliki kelebihan dan kekurangannya masing-masing. Beberapa *framework* yang dibandingkan pada pembahasan ini adalah Apache Spark, Apache Flink dan Kafka Stream.

Terdapat dua pendekatan pemrosesan data yang digunakan oleh *framework-framework* tersebut yaitu *micro-batch* dan *event-driven*. *Framework* yang menggunakan *micro-batch* mengumpulkan data pada rentang waktu hitungan detik atau milidetik sebelum data diproses seperti yang dilakukan oleh *framework* Spark (Le, Q. et al., 2022). Di sisi lain, *framework* yang menggunakan *event-driven* adalah Flink dan Kafka Stream (van Dongen,

G., & Van den Poel, D., 2020) atau di dalam sumber lain dikatakan *native* (Le, Q. et al., 2022). Pemrosesan *event-driven* tidak menunggu data sebelum diproses seperti *micro-batch*, namun data yang masuk langsung diproses dan menghasilkan data yang diinginkan lebih cepat.

Kedua cara ini memiliki kelebihan dan kekurangannya masing-masing. Kelebihan yang diunggulkan dari *event-driven* adalah *latency* dan *throughput* saat melakukan pemrosesan data. Pada percobaan yang dilakukan oleh Le, Q. et al. (2022) terlihat bahwa Flink memiliki *latency* dan *throughput* yang lebih konsisten dibandingkan Spark. Selain itu, pada percobaan yang dilakukan oleh van Dongen, G. dan Van den Poel, D. (2020) terlihat bahwa Flink dan Kafka Stream memiliki *delay* yang lebih sedikit dibandingkan Spark. *Delay* yang dimiliki Spark disebabkan oleh *micro-batch* yang mengumpulkan data terlebih dahulu sebelum memprosesnya. Selain itu, *latency* yang dimiliki Flink saat melakukan proses *join* juga lebih rendah daripada Spark. Dengan demikian, kelebihan yang ditawarkan oleh *framework* yang dengan cara *event-driven* yaitu kestabilan dan kecepatan dalam melakukan pemrosesan data.

Di sisi lain, Spark memiliki kelebihan yaitu mempunyai *latency* yang sedikit lebih rendah dibandingkan Flink saat melakukan *windowing* seperti yang ditunjukkan pada percobaan van Dongen, G. dan Van den Poel, D. (2020). Hal yang sama juga ditunjukkan pada percobaan Le, Q. et al. (2022), namun pada penelitian ini Flink menunjukkan konsistensi *latency* dan *throughput* dibandingkan dengan Spark. *Latency* yang sedikit lebih rendah ini disebabkan karena *windowing* dan *micro-batch* memiliki konsep yang sama. Kedua proses ini mengumpulkan data walaupun dengan rentang waktu yang berbeda.

Ketahanan *latency* dan *throughput* yang tetap rendah saat terjadi pertumbuhan data disebabkan Flink memiliki sistem *backpressure* yang berbeda dengan Spark. *Backpressure* merupakan kemampuan untuk mengatur laju data ketika terjadi penumpukan data pada sistem. Sistem *backpressure* Flink lebih kompleks dibandingkan Spark sehingga dapat menangani penumpukan data lebih baik.

Dari beberapa kelebihan dan kekurangan yang sudah dijelaskan dapat disimpulkan bahwa *framework* dengan *event-driven* lebih efektif untuk sistem yang membutuhkan kecepatan. Sementara itu, *framework* dengan *micro-batch* lebih sesuai digunakan untuk sistem yang

membutuhkan akurasi data dalam pemrosesannya. Selain itu, *framework event-driven* Flink mempunyai kestabilan pemrosesan data dalam jumlah banyak. Pemrosesan data dalam jumlah banyak merupakan salah satu kasus yang dihadapi oleh arsitektur *big data* pada penelitian ini. Oleh karena itu, penelitian ini menggunakan Flink sebagai *framework* yang digunakan di *streaming layer*.

6.3.2.2 Apache Cassandra

Acquisition layer merupakan *layer* yang berfungsi untuk menyediakan data yang diproses di *streaming layer*. Seperti yang ditunjukkan oleh Gambar 6.2, *acquisition layer* memiliki Cassandra yang menyimpan hasil *Change Data Capture* (CDC). Data yang disimpan oleh Cassandra digunakan untuk melengkapi data *stream* yang diproses di *streaming layer*. Oleh karena itu, untuk mengimbangi kecepatan dari pemrosesan data *stream* yang dilakukan *streaming layer*, diperlukan kemampuan membaca data yang cepat dari *database* yang menyimpan hasil CDC tersebut. Selain itu *database* dapat diakses dengan mudah dan selalu tersedia.

Database yang memiliki sifat selalu tersedia (*availability*) yang terjamin adalah *database* NoSQL. *Database* NoSQL memiliki beberapa jenis yaitu *key-value*, *document*, *column* dan *graph*. Jenis *graph* tidak cocok untuk diimplementasikan karena data hasil CDC tidak membentuk suatu *graph*. Selain itu, jenis *key-value* tidak cocok karena perubahan data yang ada memiliki lebih dari satu *key* dan *value*. *Database* jenis ini lebih sesuai digunakan sebagai *database* sementara dalam bentuk *cache*. Jenis *database* berbentuk *document* juga tidak sesuai untuk diimplementasi karena *database* jenis ini diterapkan saat menyimpan data yang berbentuk *unstructured*. Berbeda dengan data hasil CDC yang berbentuk *structured*. Oleh karena itu, *database* jenis *coulumn* sesuai untuk diterapkan pada penelitian ini. Terdapat beberapa pilihan untuk *database* ini yaitu Cassandra dan HBase.

Pada penelitian yang dilakukan Ansari, M.H. et al. (2019) dilakukan pengujian terhadap beberapa *database* yang digunakan untuk mengumpulkan data dari data generator *Air Sensor* untuk dianalisis menggunakan Spark. Penelitian tersebut menguji beberapa jenis *database* yaitu ElasticSearch, MongoDB, HBase dan Cassandra. Hasilnya adalah Cassandra memiliki *latency* rendah dan *throughput* yang tinggi pada proses membaca dan

menulis data. Oleh karena itu, kemampuan Cassandra dengan membaca dan menulis yang lebih cepat diantara *framework* yang lain Cassandra dipilih untuk diterapkan pada penelitian ini. Cassandra juga merupakan *database* yang terdistribusi sehingga data tetap dapat diakses walaupun mengalami kesalahan pada *database*.

6.3.2.3 InfluxDB

Selain menggunakan *database* NoSQL jenis *column*, arsitektur ini juga menggunakan jenis *database* NoSQL lain yaitu *time-series*. Jenis *database* ini dipilih untuk diimplementasikan pada *serve layer* untuk menyimpan data yang digunakan untuk visualisasi berbentuk *dashboard* yang memiliki rentang waktu. Terdapat beberapa *time-series database* yang sudah diuji pada penelitian Hao, Y. et al. (2021) yaitu InfluxDB, TimeScaleDB, Druid dan OpenTSDB.

Masing-masing *database* ini memiliki algoritma tersendiri untuk mengatur penyimpanan data. InfluxDB menggunakan *Time Structure Merge Tree* (TSM) sebagai algoritmanya, TimeScaleDB dibangun dari PostgreSQL sementara Druid merupakan *open source* untuk OLAP serta OpenTSDB yang dibangun dari HBase.

Strategi penyimpanan setiap *database* tersebut berbeda-beda. Saat menyimpan data, tidak semua *database* tersebut melakukan kompresi untuk menghemat tempat penyimpanan. Semakin kecil rasio kompresi maka semakin efektif *database* tersebut dalam menyimpan data. Penelitian yang dilakukan oleh Hao, Y. et al. (2021) menunjukkan InfluxDB memiliki rasio paling kecil dalam melakukan kompresi, sedangkan OpenTSDB tidak melakukan kompresi sehingga InfluxDB menyimpan data paling efektif dibandingkan *database* lainnya.

Selain dapat menyimpan data secara efektif, Hao, Y. et al. (2021) menunjukkan bahwa InfluxDB memiliki *throughput* yang stabil. Hal ini ditunjukkan dengan *throughput* mengalami konsistensi peningkatan dari percobaan injeksi data yang terus meningkat. InfluxDB dapat melakukan hal tersebut karena memiliki algoritma *Time Structure Merge Tree* (TSM) dalam penyimpanan data. *Database* lainnya dalam penelitian Hao, Y. et al. (2021) memiliki performa yang naik turun ketika diuji dengan hal yang sama.

Pada penelitian lain yang dilakukan oleh Praschl, C., et al. (2022) menunjukkan bahwa InfluxDB memiliki *throughput* dan *read time* yang lebih baik dibandingkan dengan *database* jenis lain seperti PostgreSQL dan MongoDB untuk menyimpan data dalam bentuk *time-series*. Hal ini membuktikan bahwa algoritma TSM yang digunakan oleh InfluxDB dibuat agar InfluxDB dapat menyimpan data *time-series* dengan baik.

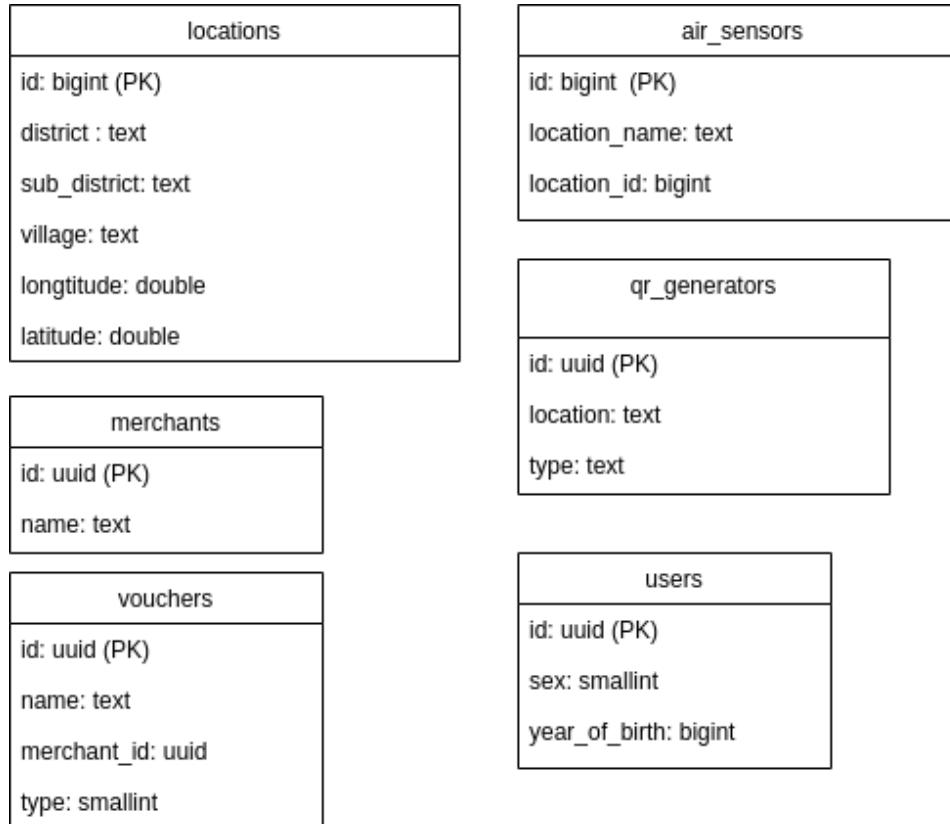
Dari hasil *benchmark* yang telah dilakukan oleh penelitian-penelitian terdahulu dapat disimpulkan bahwa InfluxDB memiliki *throughput* dan *read time* yang baik. Hal ini sesuai dengan kebutuhan dari *serve layer* pada penelitian ini. *Serve layer* menerima data *stream* yang sudah diproses oleh *streaming layer* dalam jumlah besar. Selain itu pembacaan data yang dilakukan oleh *visualization layer* yaitu Grafana juga harus memiliki kecepatan yang baik sehingga tidak terjadi *delay*.

6.3.3 Penyimpanan Data

Penyimpanan data pada arsitektur *big data* sistem aplikasi Mahoni perlu dirancang untuk mengetahui detail data-data yang disimpan pada setiap *database* yang digunakan. Rancangan *database* bertujuan agar data yang disimpan dapat mendukung visualisasi data yang menjadi tujuan dari arsitektur *big data* ini. Pada subbab berikut, dijelaskan rancangan *database* yang digunakan oleh Cassandra, InfluxDB, dan *data warehouse*.

6.3.3.1 Rancangan Database Cassandra

Database Cassandra membutuhkan skema tabel data yang disimpan sebelum digunakan. Walaupun data yang disimpan di dalam *database* ini merupakan data yang sama dengan data yang disimpan oleh *database* servis, namun Cassandra memiliki tipe data yang berbeda serta tidak mendukung konsep relasi antar tabel. Selain itu untuk efisiensi Cassandra hanya menyimpan beberapa data saja yang terkait dengan proses *enrichment* Flink seperti yang ditunjukkan pada Gambar 6.3.



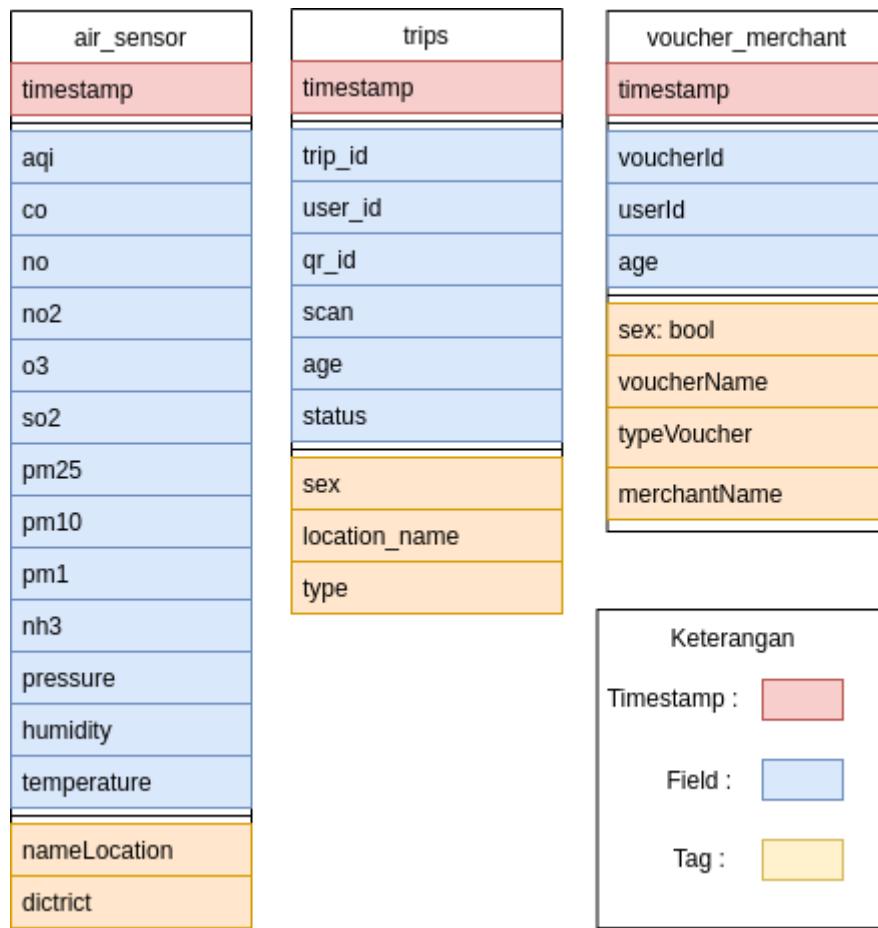
Gambar 6.3 Skema Diagram Cassandra

Cassandra menyimpan 6 tabel data non-transaksional yaitu **air_sensors**, **locations**, **vouchers**, **merchants**, **users**, dan **qr_generators**. Tabel-tabel tersebut memiliki relasinya masing-masing di dalam *database* servis, tetapi Cassandra tidak mendukung hal tersebut sehingga semua tabel terkait harus dibaca ketika membaca tabel yang berkorelasi. Cassandra menuliskan data baru berdasarkan ID yang sama. Namun ketika ID tersebut sudah ada, Cassandra melakukan *update* sehingga tidak ada data yang memiliki ID yang sama di dalam Cassandra. Hal ini mempermudah proses *enrichment* yang dilakukan *streaming layer* agar tidak membaca dua atau lebih data dengan ID yang sama.

6.3.3.2 Rancangan Database InfluxDB

Berbeda dengan Cassandra, *database* InfluxDB menyimpan data transaksional beserta *timestamp* waktu terjadinya data tersebut. Data transaksional tidak memiliki data yang lengkap sehingga membutuhkan *enrichment* dari data non-transaksional untuk mendapatkan detail yang menyeluruh pada data transaksional tersebut. Proses *enrichment*

dilakukan di dalam Flink, sehingga diperlukan skema awal dari InfluxDB sesuai dengan Gambar 6.4 untuk mengetahui data apa saja yang harus dilakukan *enrichemnt*.

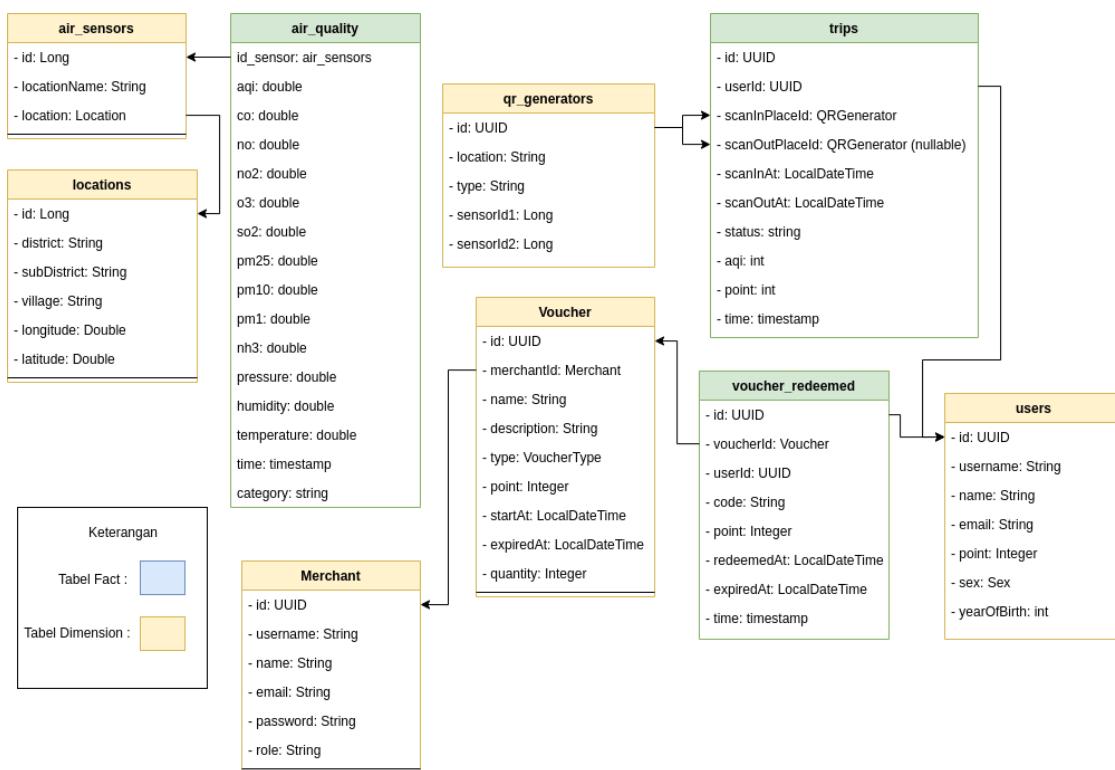


Gambar 6.4 Skema Diagram InfluxDB

Data yang disimpan di InfluxDB disimpan dalam satu *bucket* dengan tiga *measurement* yang berbeda seperti yang tergambar pada Gambar 6.4. Setiap data disimpan dengan jenis penyimpanan yang berbeda-beda sesuai dengan fungsi dan karakteristiknya. Data yang tidak unik atau memiliki perbedaan yang sedikit satu sama lain disimpan menjadi *tag*. *Tag* menjadi penyaring data saat divisualisasikan melalui *dashboard*, contohnya seperti **nameLocation** dan **district** yang menjadi penyaring data kualitas udara berdasarkan lokasi. Lalu untuk data yang unik seperti **trip_id** atau memiliki rentang yang luas **aqi** disimpan di *field*. Data tersebut merupakan data yang yang ditampilkan dan dihitung secara agregat dengan mencari nilai maksimum, minimum, dan rata-rata.

6.3.3.3 Rancangan Data Warehouse

Rancangan data pada *data warehouse* menggunakan *star schema* yang dibuat pada Gambar 6.5. *Star schema* dibuat dengan tabel *dimension* yang merupakan hasil CDC dan tabel *fact* yang merupakan data transaksional yang diproduksi oleh Kafka. Skema *data warehouse* ini merupakan gabungan dari skema yang dibuat untuk Cassandra dan InfluxDB. Skema ini menjadi “*single version of truth*” data transaksional yang ada di sistem aplikasi Mahoni.



Gambar 6.5 Skema Star Data Warehouse

Tabel *fact air_quality* memiliki data indikator kualitas udara tetapi tidak memiliki informasi mengenai lokasi dari sensor udara tersebut. Dengan adanya tabel *dimension air sensors* dan *locations*, tabel *fact trip* memiliki informasi lebih terkait *air quality* pada sensor udara tertentu. Lalu pada tabel *fact trip* hanya berisi informasi perjalanan pengguna. Tabel *dimension users* dan *qr_generators* melengkapi tabel *fact* tersebut sehingga informasi lengkap tentang pengguna dan tempat pengguna melakukan perjalanan dapat diketahui. Hal yang sama juga terjadi pada tabel *fact redeem_voucher* yang hanya

menyimpan informasi pembelian *voucher*. Dengan adanya tabel *dimension users*, *vouchers*, dan **merchants** data **redeem_voucher** menjadi lebih lengkap.

6.4 Implementasi Arsitektur *Big Data*

Implementasi arsitektur *big data* dilakukan setelah proses perancangan selesai. Semua implementasi arsitektur *big data* di-deploy pada Google Cloud Platform (GCP) dengan spesifikasi dan kebutuhan masing-masing. Pada masing-masing subbab ini dijelaskan proses *deployment* tersebut dari awal sampai akhir.

6.4.1 Implementasi *Acquisition Layer*

Acquisition layer menjadi sumber data pada arsitektur *big data* dari dua *framework* yaitu Kafka dan Cassandra. Kafka menyediakan data berdasarkan topik yang sudah ditentukan. Topik yang diolah oleh arsitektur *big data* ini merupakan topik dengan data transaksional yang terkait kualitas udara, riwayat perjalanan, riwayat pembelian *voucher* pada aplikasi Mahoni. Proses *deployment* Kafka sudah dijelaskan sebelumnya pada bab 5. Selain Kafka, Cassandra menyediakan data non-transaksional hasil CDC yang di-deploy di GCP.

Sesuai yang sudah dijelaskan pada bab 2, Cassandra dapat dibangun dengan banyak *node* yang memudahkan untuk melakukan pengembangan secara horizontal. Cassandra dibangun menggunakan tiga *node* yang dibangun di GCP dengan memanfaatkan *product* Cassandra yang sudah tersedia pada *marketplace* GCP³⁴. *Product* ini mengatur jumlah *node* yang diinginkan dan mengurus komunikasi antar *node* secara otomatis. Spesifikasi yang dimiliki oleh Cassandra setiap *node*-nya yaitu menggunakan mesin *e-2 medium* dengan memori 4 GB, penyimpanan sebesar 20GB, dan terletak pada asia-southeast2 (Jakarta) dengan *port* 9042.

```
CREATE KEYSPACE mahoni WITH replication = {'class': 'SimpleStrategy',
'replication_factor': 3};

USE mahoni;
...
```

Kode 6.1 Query Skema Cassandra

³⁴ <https://console.cloud.google.com/marketplace/product/click-to-deploy-images/cassandra>

```

...
CREATE TABLE air_sensors (
    id bigint,
    location_name text,
    location_id bigint,
    PRIMARY KEY (id)
);

CREATE TABLE locations (
    id bigint,
    district text,
    sub_district text,
    village text,
    longitude double,
    latitude double,
    PRIMARY KEY (id)
);

CREATE TABLE merchants (
    id uuid,
    name text,
    PRIMARY KEY (id)
);

CREATE TABLE vouchers (
    id uuid,
    name text,
    merchant_id uuid,
    type smallint,
    PRIMARY KEY (id)
);

CREATE TABLE qr_generators (
    id UUID,
    location text,
    type text,
    PRIMARY KEY (id)
);

CREATE TABLE users (
    id UUID,
    sex smallint,
    year_of_birth bigint,
    PRIMARY KEY (id)
);

```

Kode 6.1 Query Skema Cassandra (sambungan)

Setelah proses *deployment* telah selesai, proses selanjutnya yaitu membentuk skema tabel yang dibutuhkan serta pengaturan replikasi data. Skema yang dibuat pada Kode 6.1 sesuai

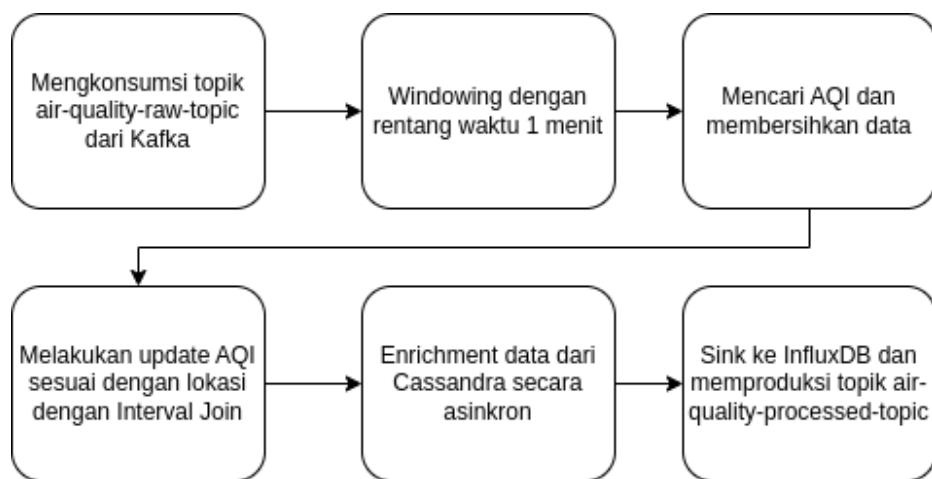
dengan skema yang telah dirancang pada Gambar 6.3. Replikasi yang digunakan pada penelitian ini sebanyak tiga *node* sesuai pada Kode 6.1 dengan mengatur ‘**replication_factor**’. Dengan replikasi tersebut, maka data disimpan pada setiap *node* yang tersedia karena jumlah *node* dengan jumlah replikasi sama. Oleh karena itu, jika satu *node* mengalami kegagalan, maka *node* lain dapat melakukan *backup* data.

6.4.2 Implementasi *Streaming layer*

Streaming layer berfungsi untuk memproses data menjadi keluaran yang diinginkan dari data-data yang tersedia pada *acquisition layer*. Pada subbab ini dijelaskan skema yang dijalankan pada setiap *job*, proses *enrichment* dan *deployment* Flink. *Job* merupakan kumpulan tugas yang dikerjaan saat memproses data. Semua sumber kode yang diimplementasikan oleh Flink, dapat diakses melalui GitHub Mahoni³⁵.

6.4.2.1 Pembuatan *Air Quality Job*

Air quality job mengonsumsi topik Kafka yaitu *air-quality-raw-topic*. Topik ini merupakan data yang diproduksi oleh data generator Air Sensor. Hasil dari topik ini ada dua yaitu memproduksi topik *air-quality-processed* yang dikonsumsi oleh *air quality service* dan melakukan *sink* ke InfluxDB dengan *measurement air_sensoes*. Untuk menghasilkan keluaran tersebut, dibutuhkan beberapa proses data seperti yang digambarkan pada Gambar 6.6.



Gambar 6.6 Alur *Air Quality Job*

³⁵ <https://github.com/Mahoni-Smart-City/mahoni-flink-job>

Skema *air quality job* diawali dengan melakukan koneksi ke Kafka untuk mengonsumsi topik *air-quality-raw-topic*. Selanjutnya dilakukan perhitungan AQI yang membutuhkan proses *windowing* di dalamnya. Perhitungan AQI membutuhkan perhitungan dari masing-masing indikator kualitas udara seperti PM10, PM25, CO, dan lain-lain. Masing-masing indikator ini sesuai dengan kaidah Air Quality Assessment Division (2018) menghitung AQI berdasarkan rentang waktu dalam waktu jam. Dalam rentang waktu tersebut, dicari rata-rata dari indikator kualitas udara yang digunakan. Lalu berdasarkan tabel konversi AQI pada Lampiran 1, ditentukan nilai AQI yang didapat dari indikator tersebut. Nilai AQI yang didapatkan setiap indikator dibandingkan untuk mencari nilai tertinggi.

Dalam proses perhitungan AQI yang sudah dijelaskan sebelumnya, penelitian ini menggunakan *tumbling windowing* yang dimiliki oleh Flink dalam rentang waktu satu menit. *Tumbling windowing* merupakan *windowing* yang menggunakan rentang waktu tertentu tanpa *overlap*. Pada penelitian ini, data yang digunakan untuk perhitungan AQI adalah PM10 dan PM25. Hal ini disebabkan data asli yang tersedia³⁶ hanya PM10 dan PM25. Sementara itu, indikator kualitas udara lain memiliki nilai yang acak dari data generator *Air Sensor* sehingga menyebabkan perhitungan AQI tidak akurat. Selama proses *windowing* dilakukan perhitungan AQI yang diimplementasi dengan Kode 6.2 sesuai dengan kaidah yang digunakan.

```
DataStream<Tuple3<String, Integer, String>> pm10 = keyedAirQualityRaw
    .window(TumblingProcessingTimeWindows.of(Time.minutes(1)))
    .process(new AqiMeasurement.AqiPM10());
public static class AqiPM10
    extends ProcessWindowFunction<AirQualityRawSchema,
    Tuple3<String, Integer, String>, String, TimeWindow>{
    @Override
    public void process(
        String key,
        Context context,
        Iterable<AirQualityRawSchema> airQuality,
        Collector<Tuple3<String, Integer, String>> out) throws Exception {
        double pm10Sum = 0.0;
        int count = 0;
    ...
}
```

Kode 6.2 Windowing Air Quality Job

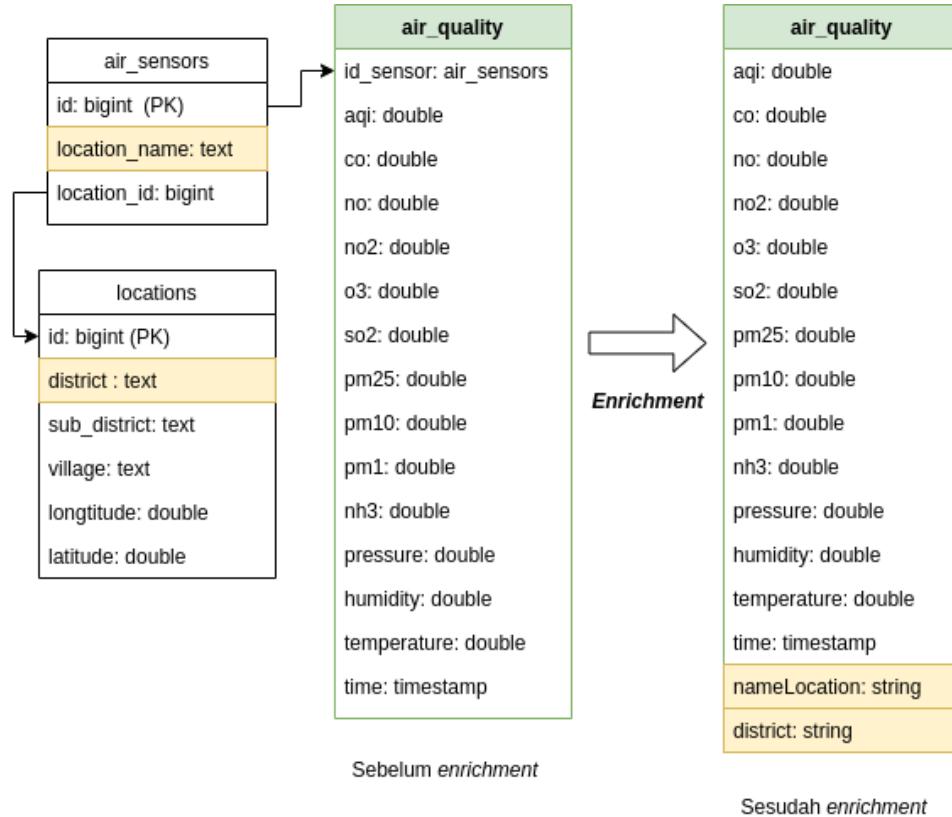
³⁶ airly.org

```
    double pm10Avg = Math.round(pm10Sum);
    Tuple2<Integer, String> aqi = new Tuple2<>();
    if (pm10Avg >= 0 && pm10Avg <= 54) {
        aqi = AqiMeasurement.calculateAqi(1,0,54, pm10Avg);
    } else if (pm10Avg >= 55 && pm10Avg <= 154) {
        aqi = AqiMeasurement.calculateAqi(2,55,154, pm10Avg);
    } else if( pm10Avg >= 155 && pm10Avg <= 254 ){
        aqi = AqiMeasurement.calculateAqi(3,155,254, pm10Avg);
    } else if (pm10Avg >= 255 && pm10Avg <= 354) {
        aqi = AqiMeasurement.calculateAqi(4,255,354, pm10Avg);
    } else if (pm10Avg >= 355 && pm10Avg <= 424) {
        aqi = AqiMeasurement.calculateAqi(5,355,424, pm10Avg);
    } else if (pm10Avg >= 425 && pm10Avg <= 504) {
        aqi = AqiMeasurement.calculateAqi(6,425,504, pm10Avg);
    } else if (pm10Avg >= 505 && pm10Avg <= 604) {
        aqi = AqiMeasurement.calculateAqi(7,505,604, pm10Avg);
    } else {
        aqi = AqiMeasurement.calculateAqi(7,505,604, pm10Avg);
    }
    out.collect(Tuple3.of(key,aqi.f0,aqi.f1));
}
}
```

Kode 6.2 Windowing Air Quality Job (sambungan)

Selanjutnya setelah AQI didapatkan, dilakukan *interval join* untuk melengkapi nilai AQI sesuai dengan kode sensor yang ada. *Interval join* dilakukan untuk menggabungkan hasil *windowing* berupa AQI dengan data *air-quality-raw-topic* dengan rentang waktu satu menit. Hal ini dilakukan agar nilai AQI yang sudah didapatkan dari satu menit sebelumnya dapat mengisi nilai AQI untuk satu menit berikutnya.

Proses yang dilakukan selanjutnya adalah *enrichment* data sensor udara. Data yang diterima dari *air-quality-raw-topic* hanya berisi data komponen udara, tidak dilengkapi dengan rinci mengenai lokasi sensor udara yang menghasilkan data tersebut. Oleh karena itu, diperlukan proses *enrichment* yang menggabungkan data *air-quality-raw-topic* dengan data tabel *air_sensors* dan *locations* yang ada di Cassandra sesuai dengan Gambar 6.7. Proses *enrichment* dilakukan secara asinkron untuk meminimalkan beban kerja yang dilakukan oleh Flink yang melakukan koneksi dan membaca data dari Cassandra sesuai dengan Kode 6.3.



Gambar 6.7 Proses *Enrichment* Data pada *Air Quality Job*

```

AirQualityRawSchema airQualityRawSchema = input.f0;
String categoryAqi = input.f1;
ResultSet airSensorDetail = session1.execute("SELECT location_id,
location_name FROM air_sensors WHERE id=" +
airQualityRawSchema.getSensorId());

Row rowAirSensorDetail = airSensorDetail.one();
if (rowAirSensorDetail != null){
    idLocation = rowAirSensorDetail.getLong("location_id");
    nameLocation = rowAirSensorDetail.getString("location_name");
}

ResultSet locationDetail = session2.execute("SELECT district, sub_district
FROM locations WHERE id=" + idLocation);

Row rowLocationDetail = locationDetail.one();
if (rowLocationDetail != null){
    district = rowLocationDetail.getString("district");

    subDistrict = rowLocationDetail.getString("sub_district");
}

```

Kode 6.3 *Enrichment Air Quality Job*

```

Point point = Point.measurement("air_sensor")
    .addTag("nameLocation",airQualityProcessedSchema.getNameLocation().toString())
    .addTag("district",airQualityProcessedSchema.getDistrict().toString())
    .addField("aqi",airQualityProcessedSchema.getAqi())
    .addField("co",airQualityProcessedSchema.getCo())
    .addField("no",airQualityProcessedSchema.getNo())
    .addField("no2",airQualityProcessedSchema.getNo2())
    .addField("o3",airQualityProcessedSchema.getO3())
    .addField("so2",airQualityProcessedSchema.getSo2())
    .addField("pm25",airQualityProcessedSchema.getPm25())
    .addField("pm10",airQualityProcessedSchema.getPm10())
    .addField("pm1",airQualityProcessedSchema.getPm1())
    .addField("nh3",airQualityProcessedSchema.getNh3())
    .addField("pressure",airQualityProcessedSchema.getPressure())
    .addField("humidity",airQualityProcessedSchema.getHumidity())
    .addField("temperature",airQualityProcessedSchema.getTemperature())
    .time(airQualityProcessedSchema.getTimestamp(), WritePrecision.MS);
writeApi.writePoint(point);

```

Kode 6.4 Sink InfluxDB Air Quality Job

Ketika data dari proses *enrichment* sudah selesai, selanjutnya dilakukan *sink* ke Kafka dengan topik *air-quality-processed-topic* untuk dikonsumsi oleh *Air Quality Service* dan InfluxDB sebagai *database* untuk visualisasi data. Proses *sink* merupakan proses untuk menempatkan data sesuai tujuan akhir yang diinginkan. InfluxDB tidak mengatur skema tabel saat proses *deployment*, sehingga pengaturan bentuk data diatur pada Kode 6.4 sesuai yang sudah direncanakan pada Gambar 6.4. Lalu untuk proses produksi data ke Kafka, dilakukan serialisasi terlebih dahulu dalam bentuk Avro sesuai dengan skema Avro yang sudah dibuat.

6.4.2.2 Pembuatan Trip Job dan Merchant & Voucher Job

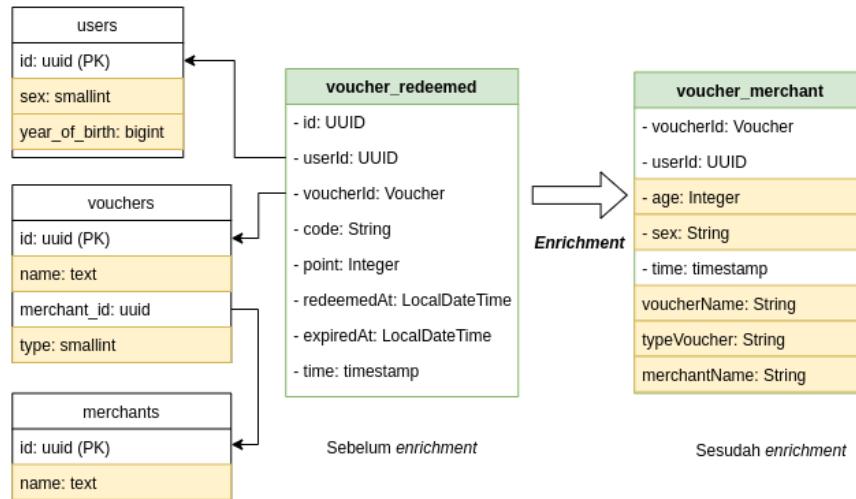
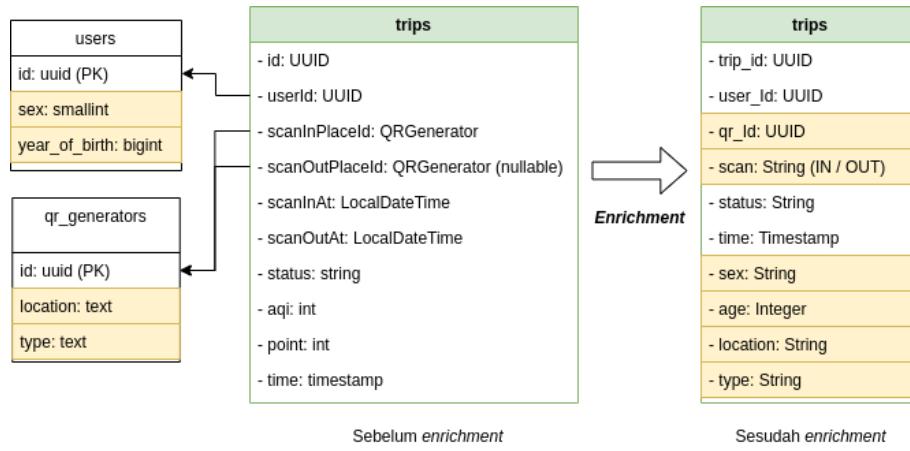
Berbeda dengan *air quality job*, *trip job* dan *merchant & voucher job* tidak melakukan banyak proses untuk memproses data. Proses yang dilakukan hanya *enrichment* dan *sink* ke InfluxDB seperti yang ditunjukkan Gambar 6.8. Proses data lain tidak diperlukan karena data yang dikonsumsi merupakan data pengguna yang tidak perlu diproses lebih jauh lagi.



(a) Alur job trip



(b) Alur job voucher & merchant

Gambar 6.8 Alur Trip Job dan Voucher & Merchant Job**Gambar 6.9** Proses Enrichment Trip Job dan Voucher & Merchant Job

Proses *enrichment* dilakukan secara asinkron untuk meminimalkan beban kerja Flink dalam koneksi dan membaca data Cassandra. *Trip job* melakukan *enrichment* untuk melengkapi informasi perjalanan seperti lokasi, jenis kegiatan (*scan in* atau *scan out*), tipe transportasi umum, dan informasi pengguna seperti umur dan jenis kelamin sesuai dengan Gambar 6.9. Lalu untuk *voucher & merchant job* juga melakukan *enrichment* untuk melengkapi informasi tipe *voucher*, nama *merchant*, dan nama *voucher* serta informasi pengguna yang sama seperti *job trip*.

```
Point point = Point.measurement("trips")
    .addTag("sex",tripEnrichment.getSex().toString())
    .addTag("location_name",tripEnrichment.getLocationName().toString())
    .addTag("type",tripEnrichment.getType().toString())
    .addField("trip_id",tripEnrichment.getTripId().toString())
    .addField("age",tripEnrichment.getAge())
    .addField("user_id",tripEnrichment.getTripId().toString())
    .addField("qr_id",tripEnrichment.getQrId().toString())
    .addField("scan",tripEnrichment.getScan().toString())
    .addField("status",tripEnrichment.getStatus().toString())
    .time(tripEnrichment.getTimestamp(), WritePrecision.MS);
writeApi.writePoint(point);
```

Kode 6.5 Sink InfluxDB Trip Job

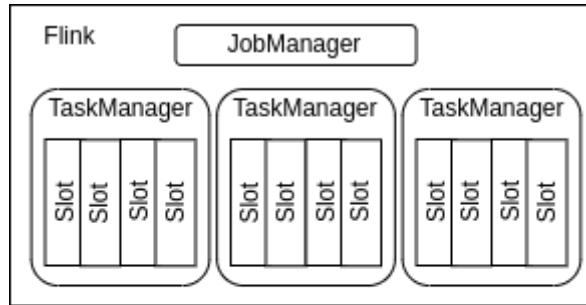
Setelah proses *enrichment* selesai, data lalu dimasukkan ke dalam InfluxDB melalui proses *sink* sesuai dengan yang sudah dirancang sebelumnya pada Gambar 6.4. Data dimasukkan sesuai dengan *measurement*, *field* dan *tag* masing-masing sesuai dengan hasil rancangan yang diimplementasi dengan Kode 6.5.

6.4.2.3 Deployment Flink

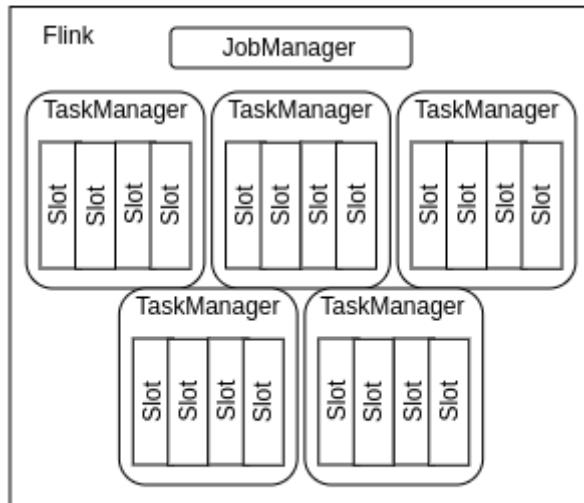
Semua *job* dibuat menggunakan bahasa pemrograman Java 11. Flink memproses *job* tersebut dalam bentuk *file JAR* yang di-*submit* menggunakan *web interface* Flink. Pada penelitian ini, Flink di-*deploy* menggunakan Kubernetes Engine pada GCP dengan lokasi *server* asia-southeast2 (Jakarta). *Deployment* Kubernetes Engine menggunakan versi otomatis sehingga hanya perlu menjalankan *file yaml* konfigurasi JobManager dan TaskManager yang bersumber dari *tutorial Flink*³⁷. Semua *file* tersebut dapat dilihat pada Lampiran 12. *Machine* yang disediakan oleh Kubenetes Engine adalah memori 2 GB

³⁷ <https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/deployment/resource-providers/standalone/kubernetes/>

dengan virtual CPU 0.5. *Deployment* yang dilakukan seperti yang terlihat pada Gambar 6.10 yaitu membangun Flink dengan 1 JobManager + 3 TaskManager, lalu 3 TaskManager tersebut akan di-*scaling* menjadi 5 TaskManager untuk kebutuhan evaluasi. Setiap TaskManager memiliki 4 *slot*. *Slot* ini berfungsi untuk melakukan *task* pada suatu *job* secara paralel.



a. Skenario 1-4



b. Skenario 5-8

Gambar 6.10 Deployment Flink

Flink menyediakan *web interface* agar dapat berinteraksi secara langsung. *Web interface* ini dapat diakses melalui port 8081. Interaksi yang dapat dilakukan yaitu melakukan pengumpulan file JAR yang berisi *job* yang sudah dibuat, informasi mengenai JobManager dan TaskManager, dan informasi mengenai *job* yang sedang dijalankan. Informasi-informasi ini berguna untuk melakukan pemantauan jalannya *job* dan memberitahuan jika terjadi eror pada *job*.

6.4.3 Implementasi *Serve Layer*

Deployment InfluxDB menggunakan *file docker-compose* yang di-deploy pada GCP menggunakan produk Virtual Machine Docker Engine Community untuk Ubuntu 20.04 pada *marketplace* GCP³⁸. Spesifikasi yang digunakan yaitu mesin *e2-medium* dengan 1-2 vCPU dan *memory* 4 GB yang dilengkapi dengan penyimpanan sebesar 10GB. *Virtual machine* berada di *asia-southeast2* (Jakarta) yang sudah dilengkapi dengan Docker di dalamnya sehingga dapat langsung membuat *file docker-compose.yml* sesuai dengan Kode 6.6 untuk dieksekusi. InfluxDB memiliki *web interface* yang dapat diakses pada *port* 8086.

```
version: "3.3"

services:
  influxdb:
    image: influxdb:2.0
    ports:
      - "8086:8086"
    volumes:
      - ./influxdb:/var/lib/influxdb
    environment:
      - INFLUXDB_ADMIN_USER=admin
      - INFLUXDB_ADMIN_PASSWORD=password
```

Kode 6.6 Docker-compose InfluxDB

Setelah proses *deployment* selesai, dilakukan proses pembuatan akun dan pemberian nama organisasi serta *bucket* yang digunakan. *Bucket* yang dibuat di dalam penelitian ini hanya satu yaitu “*mahoni_analysis*” dan di dalam *bucket* ini terdapat tiga *measurement* sesuai dengan yang sudah dirancang pada Gambar 6.4. *Bucket* yang digunakan memiliki *restriction* penghapusan data yang diatur 90 hari. InfluxDB menggunakan *Flux Query* untuk proses *query* yang tersedia pada versi InfluxDB yang digunakan yaitu versi 2.x. *Query* ini digunakan untuk mengambil data yang dilakukan oleh *visualization layer*.

6.4.4 Implementasi *Visualization Layer*

Sama seperti InfluxDB, *deployment* Grafana menggunakan produk dengan spesifikasi yang sama namun dengan *virtual machine* yang terpisah dan letak server yang berbeda

³⁸ <https://console.cloud.google.com/marketplace/product/cloud-infrastructure-services/docker-ubuntu-20>

yaitu *asia-southeast1* (Singapura). Pemisahan ini bertujuan untuk mengurangi beban masing-masing *framework* ketika menyimpan data. Untuk dapat mengakses Grafana menggunakan *port* 3000 sesuai dengan Kode 6.7.

```
version: "3.3"

services:
  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    ports:
      - "3000:3000"
```

Kode 6.7 Docker-compose Grafana

Setelah *deployment* selesai, Grafana membutuhkan *connection* yang sudah tersedia untuk tersambung dengan InfluxDB. Setelah tersambung dengan InfluxDB, pembuatan *dashboard* dilakukan dengan menggunakan *Flux Query* mengikuti versi yang dimiliki oleh InfluxDB. Grafana memiliki berbagai macam jenis visualisasi data yang dapat digunakan dan memiliki kemampuan untuk melakukan filtrasi sesuai data yang ingin ditunjukkan. Selain itu, Grafana terus tersambung ke InfluxDB selama *connection* tersedia, sehingga untuk mendapatkan data terkini perlu mengatur rentang waktu yang diinginkan dan melakukan *refresh dashboard*.

6.4.5 Implementasi *Data Warehouse*

Data warehouse diimplementasikan menggunakan produk yang ada di GCP yaitu BigQuery. Terdapat empat *dataset* yang disimpan pada BigQuery ini yaitu *air_quality*, *trip*, *user*, dan *voucher_merchant*. Seperti yang sudah dijelaskan sebelumnya bahwa *data warehouse* ini akan berfungsi sebagai “*single version of truth*”. Data yang berasal dari hasil CDC dan data transaksional dari Kafka disimpan menggunakan koneksi yang dibuat menggunakan Kafka Connect. Koneksi yang dibuat menggunakan *file* konfigurasi koneksi yang disimpan di dalam Kafka Connect seperti contoh kode konfigurasi Kode 6.8. Kafka Connect langsung menghubungkan BigQuery dengan sumber data sesuai dengan konfigurasi yang sudah diatur. Dalam pengaturan ini, BigQuery langsung membuat tabel secara otomatis sehingga tidak perlu membuat skema dan tabel terlebih dahulu.

```
{
  "name": "air-quality-processed-kcbq",
  "config": {
    "connector.class":
"com.wepay.kafka.connect.bigquery.BigQuerySinkConnector",
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "tasks.max" : "1",
    "topics" : "air-quality-processed-topic",
    "sanitizeTopics" : "true",
    "autoCreateTables" : "true",
    "allowNewBigQueryFields" : "true",
    "allowBigQueryRequiredFieldRelaxation" : "true",
    "schemaRetriever" :
"com.wepay.kafka.connect.bigquery.retrieve.IdentitySchemaRetriever",
    "project" : "mahoni-387706",
    "defaultDataset" : "air_quality",
    "keyfile" : "/home/appuser/key/mahoni-387706-969252dea32e.json"
  }
}
```

Kode 6.8 Konfigurasi Konektor Topik *air-quality-processed-topic* dengan BigQuery

Bentuk tabel yang diimplementasikan pada BigQuery menggunakan *star schema* yang berisi tabel *dimensional* yang merupakan tabel hasil CDC dan tabel *fact* merupakan data transaksional yang berasal dari Kafka. Penyimpanan BigQuery tidak menggunakan skema apa pun karena fokus utama dari BigQuery adalah mengolah data besar dengan cepat. Sehingga di dalam implementasi *data warehouse* ini tidak ada relasi data antara tabel *dimensional* dan *fact* seperti yang digambarkan oleh *star schema* seperti yang ditunjukkan Gambar 6.11. Walaupun tidak diimplementasi sesuai dengan rencana awal, pembagian tabel pada setiap *dataset* sudah sesuai dengan kebutuhan.

postgres_public_air_sensors	postgres_public_locations	postgres_public_qr_generators	postgres_public_vouchers
<ul style="list-style-type: none"> - id: integer - location_name: String - location_id: integer 	<ul style="list-style-type: none"> - id: integer - district: String - subDistrict: String - village: String - longitude: float - latitude: float 	<ul style="list-style-type: none"> - id: string - location: String - type: String - sensorId1: integer - sensorId2: integer 	<ul style="list-style-type: none"> - id: string - merchantId: string - name: String - description: String - type: integer - point: Integer - startAt: timestamp - expiredAt: timestamp - quantity: Integer
air_quality_processed_topic			voucher_redeemed_topic
<ul style="list-style-type: none"> sensorId: integer eventId: string timestamp: integer aqi: float co: float no: float no2: float o3: float so2: float pm25: float pm10: float pm1: float nh3: float pressure: float humidity: float temperature: float category: string 	<ul style="list-style-type: none"> - eventid: string - timestamp: integer - tripId: string - userId: string - scanInPlaceId: string - scanOutPlaceId: string (nullable) - scanInTimestamp: integer - scanOutTimestamp: integer - status: string - aqi: float - point: integer 	<ul style="list-style-type: none"> - eventId: string - voucherId: string - timestamp: integer - userId: string - code: String - point: Integer - redeemedAt: integer - expiredAt: integer 	<ul style="list-style-type: none"> - id: string - username: String - name: String - email: String - password: String - role: String
trip_topic			postgres_public_users
			<ul style="list-style-type: none"> - id: string - username: String - name: String - email: String - point: Integer - sex: integer - yearOfBirth: integer

Gambar 6.11 Tabel pada BigQuery

Keluaran yang dihasilkan dari *data warehouse* ini selain sebagai penyimpanan semua data adalah sebagai sumber data pembuatan laporan. Pembuatan laporan merupakan salah satu keluaran yang diharapkan oleh pihak dinas. Laporan bulanan di dalam penelitian ini dibuat menggunakan Looker Studio dari Google. *Tools* tersebut digunakan agar data yang disimpan di dalam BigQuery dapat tersambung secara langsung ke laporan karena berada di dalam ruang lingkup yang sama.

6.5 Evaluasi Arsitektur *Big Data*

Arsitektur *big data* yang sudah diimplementasikan dilakukan evaluasi dengan melakukan pengujian sesuai dengan skenario dan matriks yang sudah ditentukan. Hasil dari evaluasi ini digunakan untuk menjawab pertanyaan penelitian yang diajukan mengenai arsitektur *big data*.

6.5.1 Evaluasi Fungsionalitas

Evaluasi fungsionalitas dilakukan untuk memastikan bahwa setiap *framework* yang digunakan pada setiap *layer* arsitektur sudah berfungsi dan terkoneksi dengan baik. Terdapat beberapa hal yang dilakukan untuk mengevaluasi fungsionalitas yaitu memastikan kesesuaian antara data yang masuk dan diproses dan pengujian dengan menggunakan *integration test* pada Flink. Pengujian untuk memastikan fungsionalitas *acquisition layer* dengan *streaming layer* dilakukan dengan mengecek kesesuaian data yang diproduksi oleh Kafka dengan yang dikonsumsi oleh Flink. Pengujian ini tidak dapat dilakukan dengan *integration test* karena Flink yang mengonsumsi data dari Kafka terus membuka koneksi sehingga *integration test* tidak dapat dilakukan. Hasil pengujian yang dapat dilihat pada Gambar 6.12 menunjukkan bahwa semua topik Kafka yang dikonsumsi oleh Flink yaitu *air-quality-raw-topic*, *trip-topic*, dan *voucher-redeemed-topic* terkoneksi dengan baik.

(a) Job Air Quality

```
"eventId": "752fc213-fb3f-4fef-8cf2-b3d9cda49724",
"sensorId": "42975",
"timestamp": 1684000800000,
1> {"eventId": "752fc213-fb3f-4fef-8cf2-b3d9cda49724", "sensorId": "42975"
```

(b) Job Trip

```
"eventId": "520ba3ef-0c9c-4134-9598-171fff0dd630",
"timestamp": 1686187908599,
3> {"eventId": "520ba3ef-0c9c-4134-9598-171fff0dd630", "timestamp": 1686187908599,
```

(c) Job Voucher & merchant

```
"eventId": "b1154de4-436e-4ff6-b422-4594ddb25398",
"timestamp": 1686188314523,
1> {"eventId": "b1154de4-436e-4ff6-b422-4594ddb25398", "timestamp": 1686188314523,
```

Gambar 6.12 Kompilasi Koneksi Topik Kafka

Streaming layer dengan *serving layer* diuji fungsionalitasnya dengan menggunakan *integration test* yang tersedia pada Flink. Skenario yang dibuat adalah dengan membuat asumsi bahwa data dari Kafka sudah tersedia. Selanjutnya dilakukan pemrosesan data yang sama dilakukan pada Flink sampai proses *sink*. Setelah proses *sink* diproses dan data masuk ke dalam InfluxDB, lalu dilakukan pengecekan dengan membaca data tersebut seperti pada contoh Kode 6.9. Semua *job* pada Flink yaitu *Air Quality Job*, *Trip Job*, dan *Voucher & Merchant Job* berhasil melakukan *integration test* seperti pada Gambar 6.13

sehingga *streaming layer* dengan *serving layer* terhubung dan memiliki fungsionalitas yang baik.

```

@Test
public void EndToEndTripJob() throws Exception{
    StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
    DataStream<TripSchema> trip = env.fromElements(tripFromKafka);
    DataStream<TripEnrichment> tripEnrichment =
AsyncDataStream.orderedWait(trip, new Enrichment(), 2000,
TimeUnit.MILLISECONDS, 1000);
    tripEnrichment.addSink(new CollectSink());
    tripEnrichment.addSink(new SinkInflux());
    env.execute("test");
    System.out.println(tripId);

    String query = "from(bucket: \"mahoni_analysis\")\n" +
        "  |> range(start: -5m)\n" +
        "  |> filter(fn: (r) => r[\"_measurement\"] == \"trips\")\n" +
        "  |> filter(fn: (r) => r[\"_field\"] == \"trip_id\" and
r[\"_value\"] == \"\" + tripId + "\" )";
    Long fluxResultCount =
influxDBClient.getQueryApi().query(query).stream().count();

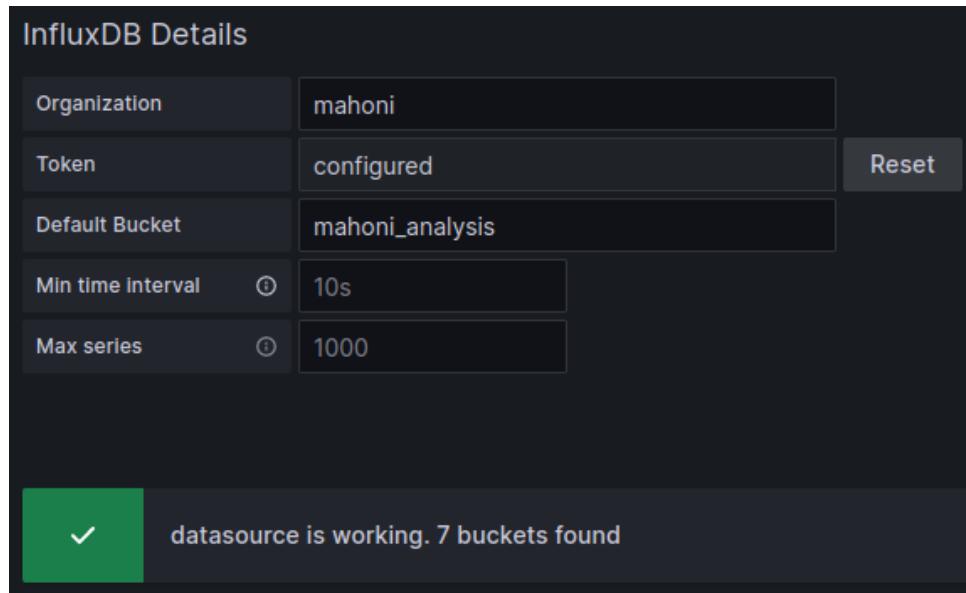
    assertEquals(tripFromKafka.getTripId(),result.getTripId());
    assertTrue(fluxResultCount>0);
}

```

Kode 6.9 *Integration Test trip-topic*

✓ ✓ AirQualityIntegrationTest	15 sec 972 ms
✓ EndToEndAirQualityJob	15 sec 972 ms
✓ ✓ TripIntegrationTest	11 sec 307 ms
✓ EndToEndTripJob	11 sec 307 ms
✓ ✓ VoucherIntegrationTest	11 sec 356 ms
✓ EndToEndVoucherJob	11 sec 356 ms

Gambar 6.13 Kompilasi *Integration Test*



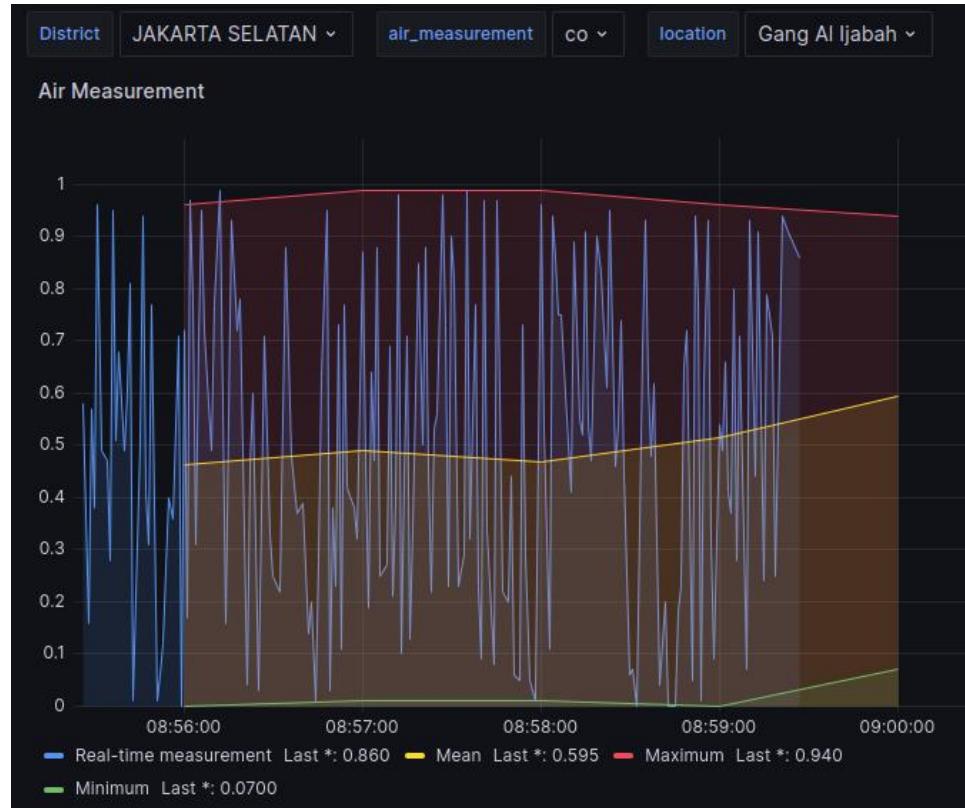
Gambar 6.14 Connection Grafana ke InfluxDB

Pengujian terakhir yaitu mengecek fungsionalitas *serve layer* dengan *visualization layer*. Pengujian ini dilakukan dengan memberikan bukti bahwa konektor InfluxDB pada Grafana memberikan tanda hijau untuk koneksi keduanya seperti pada Gambar 6.14. Selain itu, pengujian dilakukan dengan menjalankan *Flux Query* yang sama pada InfluxDB dan Grafana memberikan data yang sama sehingga dapat menampilkan *dashboard* sesuai dengan keinginan yang dijelaskan pada subbab 6.5.2.

6.5.2 Evaluasi Dashboard

Evaluasi *dashboard* dilakukan dengan mencocokkan antara *user story* kebutuhan pengguna yaitu dinas-dinas terkait di DKI Jakarta yang sudah diwawancara dengan hasil *dashboard* yang sudah dibuat. Berikut adalah *user story* dan hasil *dashboard* yang sudah diimplementasikan,

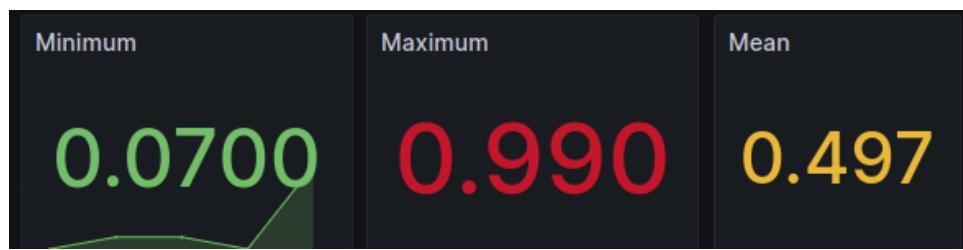
1. Dinas Lingkungan Hidup DKI Jakarta ingin mengetahui kondisi kualitas udara secara langsung berdasarkan tempat sensor udara.



Gambar 6.15 Dashboard Grafik Kualitas Udara

Pada Gambar 6.15 di atas, pengguna yaitu pihak dinas dapat melihat grafik berdasarkan waktu mengenai kondisi kualitas udara di DKI Jakarta. Terdapat empat grafik di dalam satu panel tersebut yaitu grafik perubahan secara *real-time*, grafik rata-rata, grafik nilai maksimum, dan grafik nilai minimum. Keempat grafik ini mempermudah pihak dinas untuk mengetahui kondisi suatu wilayah tersebut dengan cepat. Selain itu, grafik ini dapat dipilih berdasarkan lokasi kota (*district*) dan jalan tempat sensor udara tersebut diletakkan.

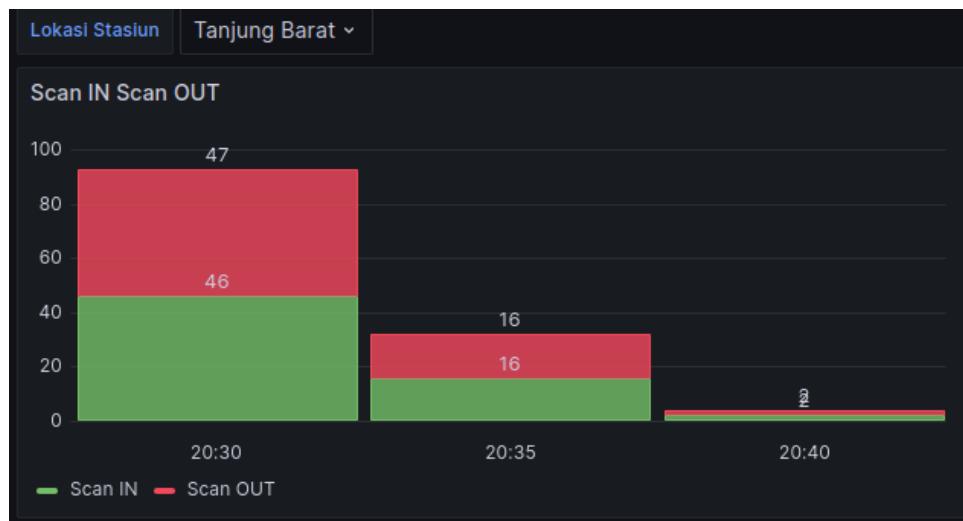
2. Dinas Lingkungan Hidup DKI Jakarta ingin mengetahui gambaran umum (maksimum, minimum, dan rata-rata) kondisi kualitas udara secara langsung berdasarkan tempat sensor udara.



Gambar 6.16 Dashboard Nilai Maksimum, Minimum, dan Rata-Rata Kualitas Udara

Pada Gambar 6.16 mempermudah pihak dinas untuk mengetahui nilai maksimum, minimum, dan rata-rata kondisi kualitas udara tanpa melihat grafik. Nilai-nilai ini merupakan hasil akumulasi data kualitas udara selama satu menit. Pada panel-panel ini, juga masih terkait dengan *dashboard* sebelumnya yang dapat dipilih berdasarkan lokasi.

3. Dinas Perhubungan DKI Jakarta ingin mengetahui jumlah pengguna yang melakukan *scan in* dan *scan out* secara langsung pada pemberhentian transportasi umum yang ada.

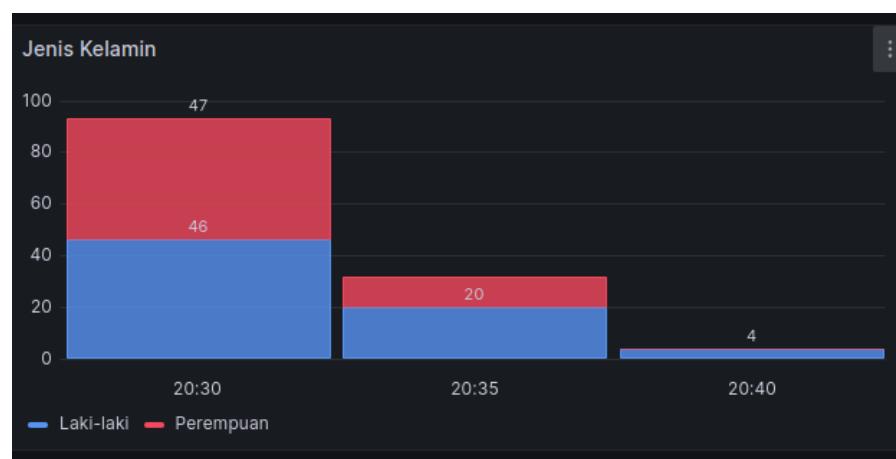


Gambar 6.17 Dashboard *Scan In* dan *Scan Out*

Gambar 6.17 menunjukkan aktivitas *scan in* dan *scan out* pengguna aplikasi Mahoni pada lokasi pemberhentian transportasi umum secara *real-time*. Grafik ini dapat membantu pihak dinas untuk menganalisis pergerakan masyarakat dalam melakukan perjalanan menggunakan transportasi umum. *Scan in* menunjukkan

titik aktivitas keberangkatan pengguna sedangkan *scan out* menunjukkan titik aktivitas kedatangan pengguna. Hal ini dapat membantu pengembangan dan analisa tempat pemberhentian transportasi umum tersebut. Panel ini dapat dipilih berdasarkan tempat pemberhentian transportasi umum.

- Dinas Perhubungan DKI Jakarta ingin mengetahui jumlah pengguna secara langsung pada suatu pemberhentian transportasi umum berdasarkan jenis kelamin pengguna.



Gambar 6.18 *Dashboard* Perjalanan berdasarkan Jenis Kelamin

Gambar 6.18 menunjukkan jumlah pengguna pada pemberhentian transportasi umum berdasarkan jenis kelamin. Dengan melihat ini, pihak dinas dapat mengetahui karakter dari pengguna transportasi umum. Sehingga pihak dinas dapat memberikan kenyamanan yang lebih untuk para penumpang.

- Dinas Perhubungan DKI Jakarta ingin mengetahui secara langsung total pengguna yang memulai perjalanan, menyelesaikan perjalanan, dan gagal pada aplikasi Mahoni berdasarkan pemberhentian transportasi umum.

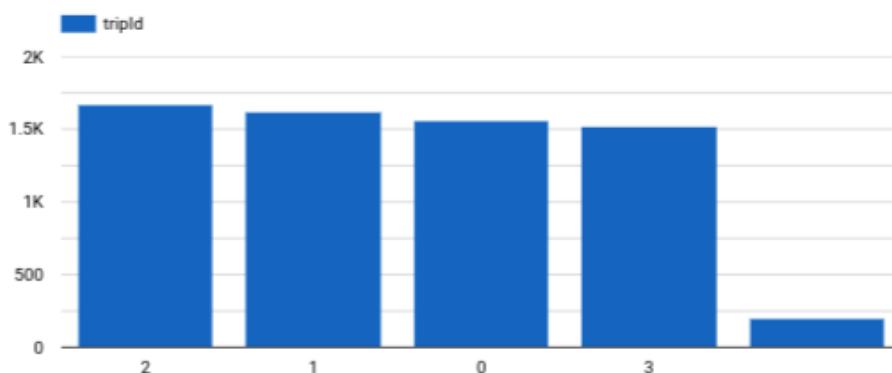


Gambar 6.19 *Dashboard* Status Perjalanan

Total pengguna berdasarkan status perjalanan yang ditunjukkan pada Gambar 6.19 merupakan hasil akumulasi satu hari pada pemberhentian tertentu. Hasil analisis dari jumlah status perjalanan ini dapat menunjukkan keberhasilan pengguna dalam menggunakan aplikasi Mahoni.

6. Dinas Perhubungan DKI Jakarta ingin mendapatkan laporan bulanan dari semua pemantauan langsung yang dilakukan.

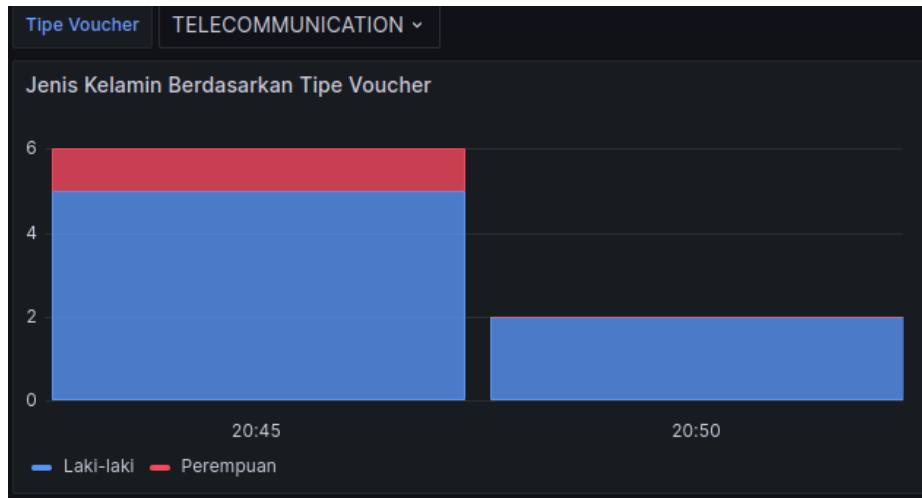
TOTAL PERJALANAN	FINISHED	ACTIVE	EXPIRED
6,574	6,524	6,574	40



Gambar 6.20 Laporan Bulanan Penggunaan Transportasi Umum

Gambar 6.20 merupakan contoh laporan bulanan yang dapat dibuat oleh pihak dinas. Sumber data laporan ini berasal dari BigQuery yang mengambil data dari 7 Mei – 7 Juni 2023. Laporan ini berisi total perjalanan, total perjalanan sesuai dengan statusnya, dan jumlah pengguna berdasarkan jenis kelaminnya.

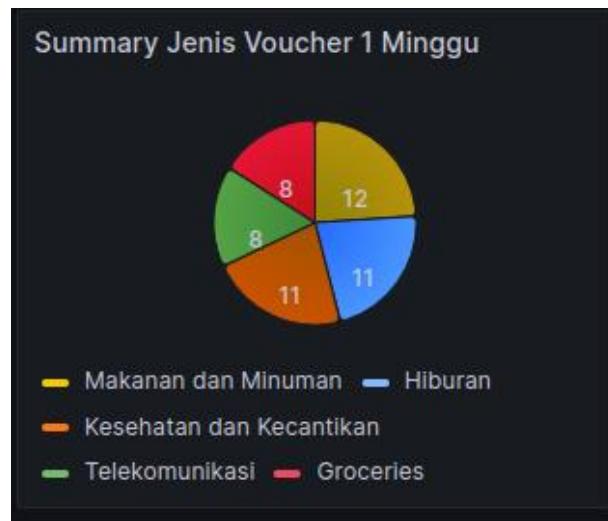
7. Dinas Pariwisata dan Ekonomi Kreatif DKI Jakarta ingin mengetahui secara langsung jumlah pengguna *voucher* pada tipe *voucher* tertentu berdasarkan jenis kelamin pengguna secara langsung.



Gambar 6.21 Dashboard Penukaran *Voucher* berdasarkan Jenis Kelamin

Pada Gambar 6.21 terlihat pembelian *voucher* berdasarkan karakteristik pengguna yaitu jenis kelamin. Panel ini melakukan penyaringan data berdasarkan tipe *voucher* yang ada. Pihak dinas dapat mendapatkan analisis tren suatu jenis produk dari panel ini berdasarkan jenis kelamin.

8. Dinas Pariwisata dan Ekonomi Kreatif DKI Jakarta ingin mengetahui jenis *voucher* yang dibeli oleh pengguna selama satu minggu.



Gambar 6.22 Dashborad Pie Chart Penukaran Voucher

Gambar 6.22 dapat menunjukkan tren masyarakat dalam rentang waktu satu minggu namun dengan tidak melihat karakteristik dari penggunanya. Pihak dinas

dapat melihat kecenderungan tertentu mengenai barang yang dikonsumsi oleh masyarakat, sehingga pihak dinas dapat melihat tren yang terjadi.

9. Dinas Pariwisata dan Ekonomi Kreatif DKI Jakarta ingin mengetahui total pembelian *voucher* dalam waktu satu minggu.



Gambar 6.23 Dashboard Total Pembelian *Voucher*

Gambar 6.23 menunjukkan total pembelian *voucher* dalam rentang waktu satu minggu. Hal ini berguna untuk mendeteksi peningkatan dan penurunan pengguna yang menukarkan *voucher* mereka.

6.5.3 Evaluasi *Scalability* dan *Reliability* Arsitektur

Pengujian *scalability* dan *reliability* pada arsitektur *big data* berfungsi untuk memastikan bahwa sistem dapat dikembangkan ketika menghadapi perubahan yang terjadi agar tidak terjadi kesalahan sistem. Pengujian ini menggunakan data sensor udara yang dibuat dari data generator *Air Sensor* yang diujikan pada Flink selama 5 menit. Flink menerima jumlah data yang semakin banyak serta dilakukan *scaling* untuk membandingkan performa. Terdapat delapan skenario yang dijalankan sesuai dengan skenario yang direncanakan pada Tabel 6.1. Pada pengujian ini dilakukan paralelisme sebanyak total *slot* yang tersedia, sehingga *job* menggunakan seluruh *node* TaskManager.

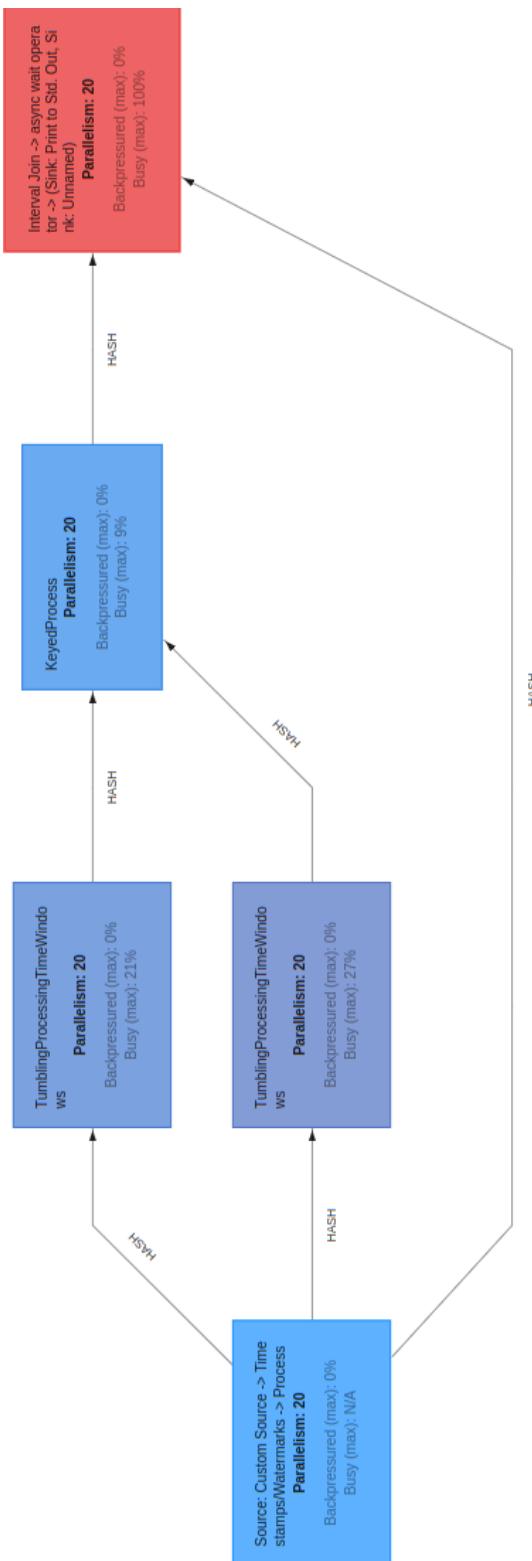
Tabel 6.2 Hasil Pengujian *Scalability* dan *Reliability*

Nomor Skenario	Total Record	Send (Proses Terakhir)	Record Receive (Proses Terakhir)	Waktu Sibuk (Proses Terakhir)	Maks. CPU JobManager	Maks. Memori JobManager	Maks. CPU TaskManager	Maks. Memori TaskManager
1	4624	4624	4624	0d	0.38	1.88	0.5	5.12
2	14464	14464	14464	1m 45d	0.03	1.95	1.139	5.19
3	23208	23208	23208	2m 36d	0.04	1.91	1.5	5.12
4	45600	39194	39194	4m 47d	0.257	1.91	1.422	5.22
5	4800	4800	4800	29d	0.237	1.83	1.3	8.47
6	14208	14208	14208	56d	0.214	1.79	1.22	9.71
7	23551	23551	23551	2m 53d	0.164	1.56	1.71	8.54
8	45760	45760	45760	3m 45d	0.233	1.92	2.17	9.56

Skenario 1—4 pada Tabel 6.3 merupakan skenario yang menggunakan 1 JobManager + 3 TaskManager dan skenario 5-8 merupakan skenario yang menggunakan 1 JobManager + 5 TaskManager. Waktu sibuk pada proses terakhir cenderung menurun pada skenario 5-8 akibat penambahan TaskManager. Selain itu, *delay* pemrosesan data pada Flink yang dialami oleh skenario 4 dapat diatasi oleh penambahan TaskManager pada skenario 8.

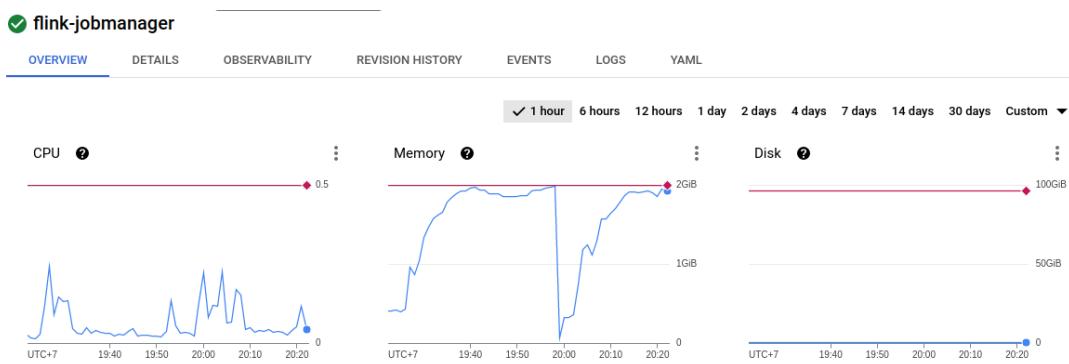
Selain berpengaruh pada berkurangnya waktu sibuk dan tidak adanya *delay* pemrosesan data pada Flink, menambah jumlah maksimum memori yang digunakan TaskManager. Hal ini disebabkan seluruh sumber daya yang dimiliki oleh TaskManager digunakan akibat paralelisme yang dilakukan untuk seluruh *slot* yang tersedia. Skenario 5-8 mempunyai TaskManager dengan total memori 10 GB dan CPU 2,5. Memori yang dipakai sekitar 90% dari total memori yang ada. Penggunaan CPU TaskManager mengalami penggunaan yang besar saat skenario 3, 4, 7, dan 8 dilakukan. Dengan batas CPU 1,5 penggunaan CPU pada skenario 3 dan 4 mencapai puncaknya, begitu pula yang dilakukan skenario 7 dan 8 yang menggunakan CPU mendekati 2,5. Hal ini menggambarkan bahwa data dengan jumlah 65 data/detik dan 130 data/detik membutuhkan *resource* yang besar untuk mengolahnya. CPU JobManager juga mengalami peningkatan pada skenario 5-8 akibat bertambahnya jumlah TaskManager yang dikendalikan. Sementara itu, memori JobManager tidak mengalami peningkatan yang signifikan walaupun penggunaannya hampir mencapai limit. Oleh karena itu,

melakukan *scaling* dengan menambah TaskManager berpengaruh pada penggunaan CPU JobManager.



Gambar 6.24 Alur Job di Web Interface Flink

Gambar 6.24 menunjukkan bahwa pada saat melakukan pemrosesan data hanya bagian akhir dari pemrosesan data yang mengalami komputasi yang tinggi. Hal tersebut menyebabkan pemrosesan data mengalami waktu sibuk pada proses tersebut. Tingginya komputasi pada proses tersebut akibat pemrosesan *interval join* untuk mencari nilai maksimal dari AQI yang telah dihitung. Selain itu juga terdapat proses *enrichment* yang dilakukan secara asinkron serta melakukan *sink* ke Kafka dan InfluxDB sehingga ketika data yang datang lebih banyak maka proses *enrichment* menjadi lebih berat.



Gambar 6.25 Kegagalan JobManager Flink

Pada proses pengujian *scalability* dan *reliability*, terdapat beberapa kendala yang hingga saat penulisan laporan ini dituliskan belum terselesaikan yaitu penggunaan memori yang banyak seperti ditunjukkan pada Gambar 6.25. Hal ini terjadi saat menjalankan *air quality job* atau terjadi anomali peningkatan secara tiba-tiba pada Kubernetes Engine. Terbatasnya jumlah memori dan CPU yang dapat *di-request* membuat *deployment* ini memiliki maksimal 0.5 CPU dan memori 2GB. Akibat lain dari keterbatasan ini adalah Flink yang *di-deploy* menggunakan Kubernetes Engine ini tidak dapat menjalankan tiga *job* sekaligus, sehingga ketika melakukan pengujian *end-to-end* harus menggunakan Flink yang dijalankan di *local* komputer.

Di samping beberapa kendala yang muncul akibat permasalahan tersebut, pengujian dapat menunjukkan bahwa Flink yang melakukan *scaling* dengan menggunakan 5 TaskManager dapat memberikan performa yang lebih baik dibandingkan menggunakan 3 TaskManager. Penambahan dua TaskManager membuat sistem tetap *reliable* dalam menjalankan sistem tanpa ada masalah. Walaupun demikian, penggunaan CPU dan memori pada TaskManager mengalami peningkatan akibat data yang terus bertambah.

Flink menggunakan semua *slot* yang diminta dengan maksimal sehingga dapat memproses data dalam jumlah banyak. Namun hal ini juga berdampak buruk karena Flink rawan menggunakan memori yang dimiliki mencapai batas penggunaan sehingga menyebabkan Flink menjadi eror.

6.6 Kesimpulan Arsitektur *Big Data*

Pada bab ini telah dilakukan proses perancangan dan implementasi arsitektur *big data* sistem aplikasi Mahoni. Perancangan arsitektur yang dimulai dengan pengumpulan data kebutuhan dinas DKI Jakarta melalui wawancara menghasilkan *user story* yang merupakan target keluaran dari arsitektur ini. Dengan mengetahui tujuan keluaran yang dihasilkan dari pemrosesan data pada arsitektur *big data* membuat proses perancangannya lebih terarah. Proses perancangan dimulai dari pemilihan bentuk arsitektur, pemrosesan yang dilakukan oleh setiap *layer*, pemilihan *framework*, dan perancangan penyimpanan data yang digunakan.

Perancangan arsitektur yang sudah dirancang dengan baik lalu diimplementasikan. Untuk membuktikan bahwa arsitektur ini sudah dirancang dan diimplementasikan dengan baik, dilakukan evaluasi arsitektur untuk menjawab pertanyaan penelitian yang ada. Evaluasi ini juga memastikan bahwa arsitektur *big data* sudah terimplementasi dengan baik.

Berdasarkan evaluasi fungsionalitas, setiap *layer* arsitektur *big data* sudah saling terhubung satu sama lain dan memproses dengan baik. Evaluasi fungsionalitas baik yang menggunakan *integration test* maupun mencocokkan data yang diproduksi dengan yang dikonsumsi oleh setiap *layer* semuanya menunjukkan keberhasilan. Keberhasilan dari evaluasi fungsionalitas ini menunjukkan bahwa *framework* yang digunakan oleh arsitektur *big data* sudah tepat. Selain itu, evaluasi dilakukan dengan mencocokkan *user story* yang dibuat dengan hasil *dashboard*. Semua *user story* sudah terpenuhi oleh *dashboard* yang dibuat sehingga *dashboard* yang dibuat sudah menunjukkan hasil yang sesuai dan akurat.

Evaluasi terakhir untuk arsitektur *big data* adalah evaluasi *scalability* dan *reliability*. Evaluasi ini bertujuan untuk menunjukkan bahwa sistem dapat berjalan dengan baik walaupun mengalami pertumbuhan data yang dapat diatasi dengan menambah komponen. Pada evaluasi ini, *scalability* dan *reliability* yang diuji adalah *framework* Flink yang

memiliki komponen JobManager dan TaskManager. Kedua komponen ini diuji dengan pertumbuhan data sensor udara oleh data generator *Air Sensor*. Hasil dari evaluasi ini adalah Flink yang dibangun dapat dilakukan *scaling* dengan menambah TaskManager sehingga Flink tetap *reliable* dalam memproses data dalam jumlah besar. Walaupun dalam pemrosesan data Flink tetap mengalami waktu sibuk pada pemrosesan data yang terakhir, tetapi *delay* pemrosesan data tidak terjadi saat melakukan *scaling* tersebut. *Delay* yang ditandai dengan berbedanya jumlah data yang datang dan diproses oleh pemrosesan data terakhir membuat komputasi pemrosesan data mengalami waktu sibuk yang lebih panjang.

Di dalam pemrosesan data ini penggunaan CPU dan memori pada TaskManager dan JobManager juga berperan penting. Flink cenderung untuk memaksimalkan penggunaan memori baik itu pada TaskManager maupun JobManager dengan menggunakan hampir 90% kapasitas maksimal. Hal itu menyebabkan ketika penggunaan memori menjadi sangat boros, memori yang digunakan dapat mencapai batas maksimal yang menyebabkan Flink mengalami eror. Untuk penggunaan CPU, TaskManager cenderung untuk memaksimalkan penggunaannya saat menghadapi data dengan jumlah yang banyak. Selain itu, melakukan *scaling* pada TaskManager membuat CPU JobManager mengalami peningkatan penggunaan. Oleh karena itu, pembuatan Flink untuk menjalankan beberapa *job* dengan jumlah data yang besar perlu memperhatikan jumlah JobManager dan TaskManager serta kemampuan masing-masing mesin seperti CPU dan memori. Melakukan *scaling* secara horizontal atau hanya menambahkan mesin yang sama tidak cukup untuk mengatasi beberapa masalah yang terjadi saat pemrosesan data, tetapi juga perlu dilakukan *scaling* secara vertikal dengan menambah kemampuan mesin.

BAB 7

EVALUASI DAN KESIMPULAN UMUM

Setelah pembahasan pengembangan sistem dan pengujinya dilakukan pada bab 4, bab 5, dan bab 6, bab ini memaparkan hasil evaluasi sistem yang dibangun secara menyeluruh dengan melakukan *end-to-end testing*. Selain itu, bab ini juga membahas kesimpulan beserta saran dari proses pekerjaan yang dilakukan.

7.1 Evaluasi Umum

Evaluasi umum dilakukan dengan menjalankan skenario pengujian *end-to-end* yang terdapat pada subbab 3.3. Proses evaluasi ini menggunakan bantuan aplikasi Postman untuk menjalankan seluruh skenario *thin-thread*. Setiap *response* yang didapatkan dievaluasi secara kuantitatif dengan melihat status kode dan statistik *error rate* serta *average response time per second* dari data yang diterima. Selanjutnya evaluasi kualitatif juga dilakukan dengan memastikan bahwa seluruh data telah masuk ke komponen *big data* dengan melihat *dashboard* dan *database big data*.

Terdapat enam langkah skenario yang dijalankan untuk menyimulasikan penggunaan aplikasi Mahoni terhadap sistem. Semua servis-servis Mahoni yang dibangun mengakomodasi jalannya pengujian, baik bersinggungan secara langsung dengan langkah skenario seperti servis pengguna, perjalanan, dan kupon dan mitra usaha, ataupun secara tidak langsung seperti servis kualitas udara. Setiap langkah skenario disusun oleh satu atau lebih *request* HTTP pada Postman. Hal tersebut dikarenakan langkah tersebut merupakan langkah umum penggunaan yang dalam proses implementasinya dapat memerlukan lebih dari satu operasi untuk setiap prosesnya.

7.1.1 Evaluasi Kuantitatif

Matriks evaluasi kuantitatif aplikasi Mahoni dilihat dengan menghitung *error rate* dan rata-rata *response time*. Untuk mengetahui *error rate* perlu ditentukan terlebih dahulu kriteria bahwa suatu skenario dianggap berhasil. Tabel 7.1 memperlihatkan kriteria kelulusan dan *endpoint* apa saja yang diperlukan untuk melakukan masing-masing skenario.

Tabel 7.1 Kriteria Kelulusan Skenario *Thin-thread*

Skenario	Endpoint untuk Menjalankan Skenario	Kriteria Kelulusan
Thin-thread		
1.	POST /api/v1/auth/register	Kode status <i>response</i> bernilai 200
2.	POST /api/v1/authenticate	Kode status <i>response</i> bernilai 200
	POST /api/v1/vouchers	Kode status <i>response</i> bernilai 200
	POST /api/v1/vouchers/detail	Kode status <i>response</i> bernilai 200
3.	POST /api/v1/users	Kode status <i>response</i> bernilai 200 Data <i>username</i> dan <i>name</i> pada <i>response</i> sesuai dengan yang ada pada <i>request</i>
4.	GET /api/v1/ /api/v1/qr-generators/[id]/generate-qr GET /api/v1/qe-generators/decode-qr POST /api/v1/trips	Kode status <i>response</i> bernilai 200 Kode status <i>response</i> bernilai 200 Kode status <i>response</i> bernilai 200 Data userId pada <i>response</i> sesuai dengan userId pada <i>request</i>
5.	POST /api/v1/redeem	Kode status <i>response</i> bernilai 200 Data status pada <i>response</i> bernilai PENDING
6.	POST /api/v1/redeem/[id]	Kode status <i>response</i> bernilai 200 Data status pada <i>response</i> bernilai REDEEMED

Seluruh validasi kelulusan skenario telah diimplementasikan pada Postman sehingga penulis hanya perlu melihat hasil yang dikeluarkan saja. Kode 7.1 memperlihatkan contoh tes pada Postman, yaitu melakukan validasi bahwa *response* yang diterima memiliki status 200 dan menghasilkan keluaran yang sesuai dengan ekspektasi.

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

pm.test("Return Correct Data", function () {
    var response = pm.response.json();
    pm.expect(response.status).to.eql("REDEEMED");
});
```

Kode 7.1 Contoh Tes pada Postman

Skenario dijalankan pada Postman menggunakan fitur *functional test*. Pada Lampiran 10 memuat *response* dari pengujian masing-masing skenario. Kemudian, pada Lampiran 11 menampilkan tangkapan gambar asli kesimpulan hasil tes dari Postman. Hasil tes tersebut kemudian dikompilasikan ke dalam Tabel 7.2.

Tabel 7.2 Hasil Skenario *End-to-End Testing*

No.	Endpoint	Status Kelulusan	Response Time
1	POST /api/v1/auth/register	✓ Kode status <i>response</i> bernilai 200	162 ms
2	POST /api/v1/authenticate	✓ Kode status <i>response</i> bernilai 200	107 ms
	POST /api/v1/vouchers	✓ Kode status <i>response</i> bernilai 200	21 ms
	POST /api/v1/vouchers/detail	✓ Kode status <i>response</i> bernilai 200	23 ms
3	POST /api/v1/users	✓ Kode status <i>response</i> bernilai 200 ✓ Data <i>username</i> dan <i>name</i> pada <i>response</i> sesuai dengan yang ada pada <i>request</i>	51 ms
4	GET /api/v1/ /api/v1/qr-generators/[id]/generate-qr GET /api/v1/qe-generators/decode-qr POST /api/v1/trips	✓ Kode status <i>response</i> bernilai 200 ✓ Kode status <i>response</i> bernilai 200 ✓ Kode status <i>response</i> bernilai 200 ✓ Data userId pada <i>response</i> sesuai dengan userId pada <i>request</i>	22 ms 15 ms 36 ms
5	POST /api/v1/redeem	✓ Kode status <i>response</i> bernilai 200 ✓ Data status pada <i>response</i> bernilai PENDING	25 ms
6	POST /api/v1/redeem/[id]	✓ Kode status <i>response</i> bernilai 200 ✓ Data status pada <i>response</i> bernilai REDEEMED	19 ms
<i>Average Response Time</i>			43 ms

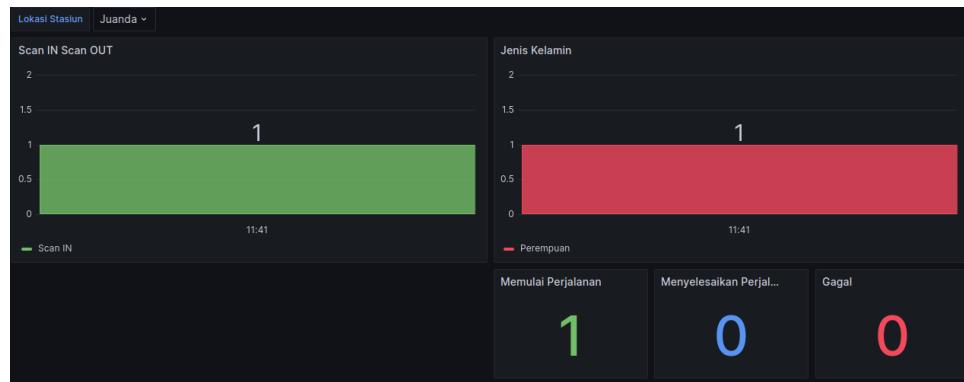
Berdasarkan tabel tersebut dapat disimpulkan bahwa seluruh *thin-thread* dalam skenario berhasil dilakukan tanpa ada eror. Dengan demikian evaluasi matriks *error rate* bernilai 0. Pada tabel juga dapat terlihat nilai *average response time* yang bernilai 43 ms.

7.1.2 Evaluasi Kualitatif

Evaluasi kualitatif dilakukan dengan memastikan bahwa seluruh data yang dibuat pada skenario *end-to-end testing* telah tersedia pada *database* dan *dashboard* arsitektur *big data*. Data pada Lampiran 10 harus dicocokkan dengan data yang terdapat pada komponen *big data* dengan melakukan pemantauan *dashboard* saat menjalankan evaluasi, kemudian mengevaluasikannya dengan kriteria kelulusan evaluasi kualitatif yang terdapat pada Tabel 7.3.

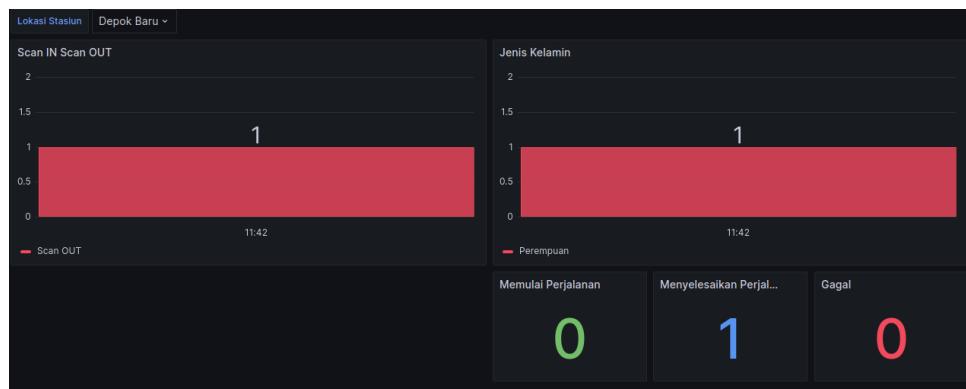
Tabel 7.3 Kriteria Kelulusan Evaluasi Kualitatif

Komponen Big Data	Kriteria Kelulusan
Dashboard	<p><i>Dashboard</i> menampilkan aktivitas <i>scan in</i></p> <p><i>Dashboard</i> menampilkan aktivitas <i>scan out</i></p> <p><i>Dashboard</i> menampilkan aktivitas penukaran <i>voucher</i> sesuai dengan kategori <i>voucher</i> dan dilengkapi dengan informasi umur serta jenis kelamin <i>user</i></p> <p><i>Dashboard</i> menampilkan nilai kualitas udara (AQI) secara <i>realtime</i></p>
Cassandra	<p>Terdapat data <i>user</i></p> <p>Terdapat data <i>merchant</i></p> <p>Terdapat data <i>voucher</i> yang dibuat oleh <i>merchant</i></p>
Google Big Query	<p>Terdapat <i>event scan in</i></p> <p>Terdapat <i>event scan out</i></p> <p>Terdapat <i>event voucher redeemed</i></p>



Gambar 7.1 Dashboard Hasil Scan In

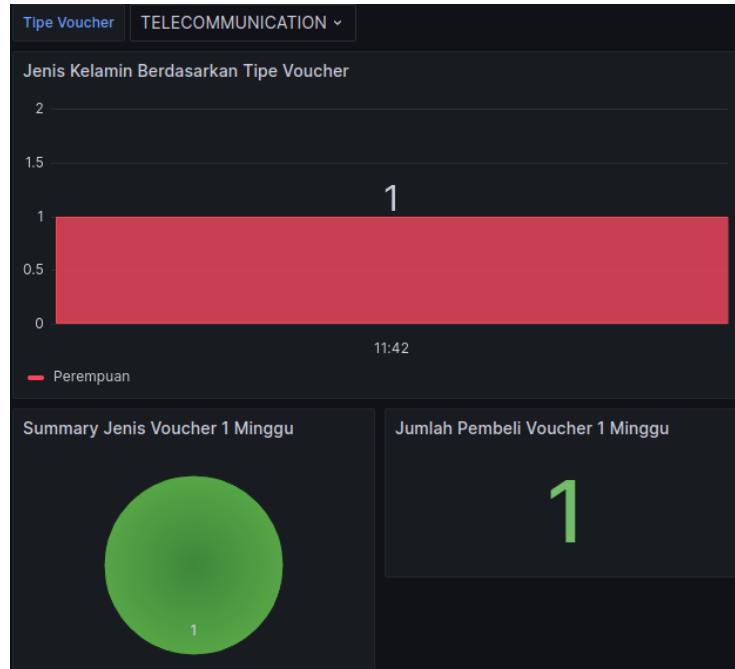
Gambar 7.1 menampilkan kondisi *dashboard* yang menunjukkan aktivitas pengguna saat melakukan *scan in*. Sesuai dengan Lampiran 10, pengguna melakukan *scan in* melalui stasiun Juanda. Pengguna berhasil melakukan *scan in* yang ditunjukkan dengan adanya pengguna yang memulai perjalanan. Pengguna yang melakukan *scan in* merupakan seorang wanita.



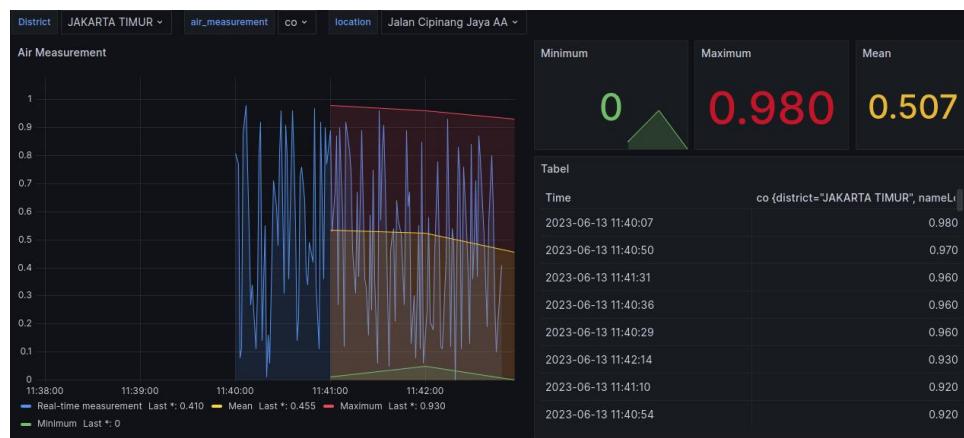
Gambar 7.2 Dashboard Hasil Scan Out

Gambar 7.2 menunjukkan bahwa pengguna tersebut melakukan perjalanan ke Depok Baru. Pengguna yang berjenis kelamin perempuan ini berhasil melakukan satu kali perjalanan dengan indikator menyelesaikan perjalanan. Hal ini sesuai dengan Lampiran 10 yang menunjukkan bahwa pengguna tersebut melakukan perjalanan dari Cawang ke Depok Baru.

Selanjutnya pada skenario ini, pengguna tersebut melakukan penukaran *voucher* dengan jenis **Telecommunication**. Pengguna tersebut membeli satu *voucher* tersebut seperti ditunjukkan pada Gambar 7.3. Pada pelaksanaan *end-to-end testing* ini dijalankan juga data generator *Air Sensor* sehingga Gambar 7.4 menampilkan pergerakan kualitas udara yang terjadi.



Gambar 7.3 Dashboard Hasil Redeemed Voucher



Gambar 7.4 Dashboard Hasil Kualitas Udara

Selanjutnya, dilakukan *query* manual pada *database* Cassandra dan Google BigQuery untuk melihat apakah data sudah diterima pada komponen tersebut. Gambar 7.5 menampilkan bahwa data terkait *merchant*, *user*, dan, *voucher* sudah tersedia di Cassandra dengan data yang sama pada Lampiran 10. Hal ini menunjukkan bahwa hasil CDC dari *database* servis berhasil disimpan oleh Cassandra.

```

    id          | merchant_id          | name           | type
    13e198db-5bb8-40b5-a442-023a965bee88 | a46602ff-0936-45a1-998b-23dd61395cad | Awesome Wooden Gloves | 3
(1 rows)
cqlsh:mahoni> select * from users where id=67eb5009-fe79-4145-839c-bde98e9afa43 allow filtering;
    id          | sex | year_of_birth
    67eb5009-fe79-4145-839c-bde98e9afa43 | 2   |      2000
(1 rows)
cqlsh:mahoni> select * from merchants where name='Greyson.Windler' allow filtering;
    id          | name
    a46602ff-0936-45a1-998b-23dd61395cad | Greyson.Windler

```

Gambar 7.5 Data pada Cassandra

a. Data Warehouse Trip

Query results				
JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
Row	eventId	timestamp	tripId	userId
1	1f7aaaaab-895a-4340-a22e-7c4...	1686631260476	d78d7079-6590-47b0-a97b-cf8...	67eb5009-fe79-4145-839c-bde...
2	6ec0c881-b630-4431-8a12-31e...	1686631260840	d78d7079-6590-47b0-a97b-cf8...	67eb5009-fe79-4145-839c-bde...

b. Data Warehouse redeemed voucher

Query results				
JOB INFORMATION		RESULTS	JSON	EXECUTION DETAILS
Row	eventId	timestamp	voucherId	userId
1	29ca4ae0-a972-466e-8971-ac...	1686631261968	13e198db-5bb8-40b5-a442-02...	67eb5009-fe79-4145-839c-bde...

Gambar 7.6 Data pada Google BigQuery

Gambar 7.6 menampilkan bahwa *event scan in*, *scan out*, dan *voucher redeemed* juga sudah diterima pada *data warehouse*. Oleh karena itu, dapat disimpulkan bahwa seluruh komponen *big data* telah berhasil diintegrasikan pada aplikasi Mahoni dan berjalan sesuai dengan masukkan yang telah diberikan pada Lampiran 10.

7.2 Kesimpulan Umum

Berangkat dari salah satu isu umum di kota besar Indonesia dan konsep aplikasi Mahoni sebagai upaya untuk mendukung pergerakan perubahan dengan mengaitkan tiga pilar dari kota cerdas, memunculkan kebutuhan dalam implementasi pengembangan sistem aplikasi Mahoni yang mampu menangani data yang kompleks dan banyak serta berbagai komponen-komponen yang saling terhubung. Adanya latar belakang tersebut membentuk

beberapa tujuan dari penelitian ini. Penelitian dan pengembangan yang dilakukan berhasil mencapai masing-masing tujuan penelitian tersebut.

1. Penulis telah merancang dan mengimplementasikan servis sebagai layanan fitur-fitur yang sesuai dengan kebutuhan pengguna untuk menggunakan aplikasi Mahoni. Perancangan fitur tersebut dilakukan dengan kegiatan *requirement gathering* yang melibatkan pengguna baik dalam pengumpulan kebutuhan hingga pengelompokan kebutuhan. Implementasi layanan fitur-fitur berupa servis dilakukan mengikuti rancangan yang telah dibuat berdasarkan translasi proses bisnis dan kebutuhan pengguna. Servis yang dibuat dapat melayani kebutuhan pengguna dan terintegrasi dengan baik kepada komponen arsitektur *event-driven*.
2. Penulis telah merancang dan mengimplementasikan arsitektur yang sesuai dengan *requirement* dari kegiatan sebelumnya. Arsitektur yang dibuat telah memenuhi kebutuhan atas *throughput* tinggi untuk mengatasi jumlah sensor yang banyak pada sistem kota cerdas dan bersifat *loosely-coupled* untuk mempermudah integrasi komponen baru pada sistem. Terlebih lagi, arsitektur yang dibuat juga dapat mengintegrasikan komponen *big data* yang membutuhkan data *stream* baik dari kegiatan *user* maupun perubahan pada *database*.
3. Penulis telah merancang dan mengimplementasikan arsitektur *big data* untuk sistem aplikasi Mahoni. Seluruh komponen arsitektur *big data* dapat berjalan dan terkoneksi dengan baik sesuai dengan fungsinya masing-masing dengan pemilihan *framework* yang tepat. Hasil yang didapatkan dari pengolahan data dapat memenuhi kebutuhan yang ada berdasarkan hasil wawancara. Selain itu, sistem terutama *framework* Flink dapat melakukan *scaling* secara horizontal untuk menghadapi pertumbuhan data baik dari data sensor maupun pengguna. Dengan demikian arsitektur yang dibuat dapat *reliable* dalam menghadapi kesalahan atau eror dalam sistem.

7.3 Saran

Pekerjaan yang dilakukan pada pengembangan dan penelitian ini merupakan tahap awal dari pengembangan aplikasi Mahoni yang sesungguhnya. Terdapat saran untuk aspek

teknis dan non teknis untuk pengembangan selanjutnya. Beberapa saran aspek teknis dari masing-masing pekerjaan adalah sebagai berikut:

1. Implementasi servis yang dilakukan terbatas untuk pemenuhan kebutuhan utama dari pengguna dalam melaksanakan proses bisnis. Adanya batasan-batasan penelitian terutama waktu menyebabkan fitur penunjang tidak dapat terimplementasi sepenuhnya. Salah satu fitur tersebut adalah fitur autentikasi dan otorisasi pengguna, yang mana baru hanya diterapkan ke salah satu servis saja. Oleh karena itu, dapat dilakukan pengembangan fitur tersebut yang terintegrasi ke semua servis. Selain itu arsitektur dan skema yang sudah dibuat pada penelitian ini dapat dicoba untuk diterapkan dengan data pengguna asli dan data sensor udara yang bersumber dari alat sensor udara.
2. Implementasi *schema registry* sebaiknya dibuat menjadi lebih dari satu *node* agar tidak terjadi *bottleneck*. Selain itu, implementasi KTable sebagai *key-value database* sebenarnya bisa digantikan oleh Redis. Hal ini dapat memberikan lebih banyak fitur pada *web service* seperti *caching*, *limiter*, dan lain-lain meskipun harus melakukan pengaturan tambahan dan tidak bersifat *fault-tolerance*.
3. Dapat dilakukan penelitian lanjutan untuk meneliti integrasi IoT dengan sistem Mahoni agar lebih efisien. Hal ini bisa dilakukan dengan menggunakan protokol lain seperti MQTT antar sensor ke sebuah *node* pengumpul data. Kemudian *node* tersebut yang mengirimkan data ke Kafka.
4. Dapat dilakukan penelitian selanjutnya terkait penggunaan *framework* yang digunakan pada arsitektur *big data*. *Framework* Flink dapat diuji lebih mendalam untuk mengetahui faktor-faktor yang mempengaruhi performa pemrosesan data. Selain itu, *framework* lain seperti Cassandra dan InfluxDB juga dapat diuji mengenai efektifitas penggunaannya. Percobaan dengan *framework* lain juga dapat dilakukan untuk mendapatkan perbandingan performa pengolahan data.
5. *Data warehouse* yang digunakan pada arsitektur *big data* dapat dikembangkan fungsi serta pengembangan yang lebih fokus kepada *data warehouse*. Penelitian yang berfokus pada pengaturan skema dan pemanfaatan lebih lanjut mengenai

data-data yang sudah dikumpulkan di dalam *data warehouse* untuk *machine learning* dan data analisis dapat menjadi topik yang menarik untuk dicoba.

Selain saran yang berkaitan dengan aspek teknis, terdapat pula beberapa saran dan harapan untuk pengembangan dalam aspek non teknis, yaitu:

1. Pengembangan aplikasi selanjutnya dapat dilakukan dengan menerapkan *usecase* secara langsung ke suatu kota agar *requirement* memenuhi kebutuhan spesifik penduduk di kota tersebut. Selain itu, masih terdapat banyak ruang untuk dilakukan pengembangan lebih lanjut, terutama untuk implementasi aplikasi berbasis *mobile* dan web.
2. Diperlukan seseorang yang sangat paham terhadap Avro *schema* untuk dapat melakukan migrasi versi dengan baik tanpa terjadi konflik.
3. Peneliti yang ingin mengembangkan topik terkait *big data* sebaiknya merupakan seseorang sudah pernah mencoba *framework* yang akan digunakan. Sehingga waktu untuk mempelajari dan menguasai *framework* lebih mendalam tidak membutuhkan waktu yang lama.

DAFTAR PUSTAKA

- Air Quality Assessment Division. (2018). *Technical Assistance Document for the Reporting of Daily Air Quality – the Air Quality Index (AQI)* (No. 454B18007). U.S. Environmental Protection Agency. Retrieved May 21, 2023, from <https://www.airnow.gov/sites/default/files/2020-05/aqi-technical-assistance-document-sept2018.pdf>
- Ansari, M. H., Tabatab Vakili, V., & Bahrak, B. (2019). Evaluation of big data frameworks for analysis of Smart Grids. *Journal of Big Data*, 6(1)
- Apache Kafka documentation.* (n.d.). Apache Kafka. Retrieved May 28, 2023, from <https://kafka.apache.org/documentation.html#compaction>
- Atlassian. (n.d.). *What is Code Coverage?* / Atlassian. <https://www.atlassian.com/continuous-delivery/software-testing/code-coverage#:~:text=With%20that%20being%20said%20it,fairly%20low%20percent%20age%20of%20coverage.>
- Balusamy, B., R., N. A., Gandomi, A. H., & Kadry, S. (2021). Big Data: Concepts, technology and Architecture. John Wiley and Sons, Inc.
- Berger, C., Blauth, R. S. M., & Boger, D. V. (1993). Kano'S Methods for Understanding Customer-Defined Quality. *Center for Quality Management Journal*, 2(4), 3–36. <https://ci.nii.ac.jp/naid/10030507522/>
- Cassandra Basics. Apache Cassandra.* (n.d.). https://cassandra.apache.org/_/cassandra-basics.html
- Chapter 4. Hardware Sizing Recommendations - Hortonworks DataFlow.* (n.d.). Cloudera Documentation. Retrieved June 1, 2023, from https://docs.cloudera.com/HDPDocuments/HDF3/HDF-3.1.0/bk_planning-your-deployment/content/ch_hardware-sizing.html

Clemson, T. (2014, November 18). *Testing Strategies in a Microservice Architecture*. martinfowler.com. Retrieved June 2, 2023, from <https://martinfowler.com/articles/microservice-testing/>

Create a stream reactor sink connector from Apache Kafka® to Apache Cassandra®. (n.d.). Aiven documentation. Retrieved June 1, 2023, from <https://docs.aiven.io/docs/products/kafka/kafka-connect/howto/cassandra-streamreactor-sink>

Debezium Architecture. (n.d.). Debezium. Retrieved May 30, 2023, from <https://debezium.io/documentation/reference/stable/architecture.html>

Direktorat Jenderal Aplikasi Informatika Kementerian Komunikasi dan Informatika. (2021). *Guideline Masterplan Smart City: Gerakan Menuju Kota Cerdas (Smart City)*. https://sinau.kuduskab.go.id/wikiit/img/slider_folder/Materi_1656565703.pdf

Dobbelaere, P., & Esmaili, K. S. (2017). Industry Paper: Kafka versus RabbitMQ. *Proceedings of the 11th ACM International Conference on Distributed Event-Based Systems, 06/2017*. <https://doi.org/10.1145/3093742.3093908>

Farzan, R., DiMicco, J. M., Millen, D. R., Dugan, C., Geyer, W., & Brownholtz, E. A. (2008). Results from deploying a participation incentive mechanism within the enterprise. <https://doi.org/10.1145/1357054.1357145>

Feick, M., Kleer, N., and Kohn, M. (2018). Fundamentals of Real-Time Data Processing Architectures Lambda and Kappa.

Filipponi, L., Vitaletti, A., Landi, G., Memeo, V., Laura, G., & Pucci, P. (2010). Smart City: An Event Driven Architecture for Monitoring Public Spaces with Heterogeneous Sensors. *2010 Fourth International Conference on Sensor Technologies and Applications*. 10.1109/SENSORCOMM.2010.50

Flink architecture. (n.d.). <https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/concepts/flink-architecture/>

- Goh, D. H., Pe-Than, E. P. P., & Lee, C. S. (2017). Perceptions of virtual reward systems in crowdsourcing games. *Computers in Human Behavior*, 70, 365–374. <https://doi.org/10.1016/j.chb.2017.01.006>
- Hao, Y., Qin, X., Chen, Y., Li, Y., Sun, X., Tao, Y., Zhang, X., & Du, X. (2021). TS-benchmark: A benchmark for time series databases. *2021 IEEE 37th International Conference on Data Engineering (ICDE)*.
- Hashem, I. A., Chang, V., Anuar, N. B., Adewole, K., Yaqoob, I., Gani, A., Ahmed, E., & Chiroma, H. (2016). The role of Big Data in Smart City. *International Journal of Information Management*, 36(5), 748–758. <https://doi.org/10.1016/j.ijinfomgt.2016.05.002>
- Hellinger, D. (2022, April). Designing and testing a highly available Kafka cluster on Kubernetes. Learnk8s. Retrieved June 1, 2023, from <https://learnk8s.io/kafka-ha-kubernetes>
- Huang, H., Su, D., & Wenjie, P. (2022). Novel Mobile Application System for Implementation of an Eco-Incentive Scheme. *Sustainability*, 14(5), 3055. <https://doi.org/10.3390/su14053055>
- Hwang, J., & Choi, L. (2019). Having fun while receiving rewards?: Exploration of gamification in loyalty programs for consumer loyalty. *Journal of Business Research*, 106, 365–376. <https://doi.org/10.1016/j.jbusres.2019.01.031>
- InfluxDB Data elements. InfluxDB data elements | InfluxDB OSS 2.7 Documentation. (n.d.). <https://docs.influxdata.com/influxdb/v2.7/reference/key-concepts/data-elements/>
- Kafka to Cassandra open source connector / Lenses.io Help Center.* (n.d.). Lenses Documentation. Retrieved May 30, 2023, from <https://docs.lenses.io/5.1/connectors/sinks/cassandrasinkconnector/>
- Kalipe, G. K., & Behera, R. K. (2019). Big Data Architectures: A detailed and application oriented analysis. *International Journal of Innovative Technology and Exploring Engineering*, 8(9), 2182–2190. <https://doi.org/10.35940/ijitee.h7179.078919>

- Kimball, R., & Ross, M. (2013). Data Warehousing, Business Intelligence, and Dimensional Modeling Primer. In *The Data Warehouse toolkit the Definitive Guide to Dimensional Modeling* (pp. 7–17). essay, Wiley.
- Kleppmann, M. (2016). *Making Sense of Stream Processing: The Philosophy Behind Apache Kafka and Scalable Stream Data Platforms* (1st ed.). 978-1-491-94010-5
- Kleppmann, M. (2018). Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. O'Reilly Media.
- Kumar, P. (2022). A critical evaluation of air quality index models (1960–2021). *Environmental Monitoring and Assessment*, 194(5), 324.
- Laigner, R., Kalinowski, M., Diniz, P., Barros, L., Cassino, C., Lemos, M., Arruda, D., Lifschitz, S., & Zhou, Y. (2020). From a Monolithic Big Data System to a Microservices Event-Driven Architecture. *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 10.1109/SEAA51224.2020.00045
- Le, Q., Chen, M., & Chen, W. (2022). Research on stream processing engine and benchmarking framework. *2022 IEEE International Conference on Networking, Sensing and Control (ICNSC)*.
- Lourenço, J. R., Cabral, B., Carreiro, P., Vieira, M., & Bernardino, J. (2015). Choosing the right nosql database for the job: A Quality Attribute Evaluation. *Journal of Big Data*, 2(1).
- Magnoni, L. (2015). Modern messaging for distributed sytems. *Journal of Physics: Conference Series* 608 (2015) 012038. doi:10.1088/1742-6596/608/1/012038
- Narkhede, N., Shapira, G., & Palino, T. (2017). *Kafka: The Definitive Guide*. 978-1-491-99065-0
- Neo4j. (2021, September 29). *10 Reasons Why Neo4j Is the Best Graph Database for Your Project*. Graph Database & Analytics. <https://neo4j.com/top-ten-reasons/>

PostgreSQL: About. (n.d.). The PostgreSQL Global Development Group.
<https://www.postgresql.org/about/>

Praschl, C., Pritz, S., Krauss, O., & Harrer, M. (2022). A comparison of relational, NoSQL and NewSQL database management systems for the persistence of time series data. *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*

Richards, M. (2015). *Software Architecture Patterns Understanding Common Architectural Styles and When to Use Them* (2nd ed.). O'reilly.

Richardson, C. (n.d.). *Pattern: Event-driven architecture*. Microservice Architecture. Retrieved May 30, 2023, from <https://microservices.io/patterns/data/event-driven-architecture.html>

Sanla, A., & Numnonda, T. (2019). A comparative performance of real-time Big Data Analytic Architectures. *2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC)*.
<https://doi.org/10.1109/iceiec.2019.8784580>

Sen, A., Ramamurthy, K., & Sinha, A. P. (2012). A model of data warehousing process maturity. *IEEE Transactions on Software Engineering*, 38(2), 336–353.
<https://doi.org/10.1109/tse.2011.2>

Silhavy, R., Silhavy, P., & Prokopova, Z. (2011). Requirements gathering methods in system engineering. *Recent Researches in Automatic Control*, 106–110.
<http://www.wseas.us/e-library/conferences/2011/Lanzarote/ACMOS/ACMOS-17.pdf>

Stopford, B. (2018). *Designing Event-Driven Systems Concepts and Patterns for Streaming Services with Apache Kafka*. O'reilly.

Streams Concepts. (n.d.). Confluent Documentation. Retrieved May 28, 2023, from <https://docs.confluent.io/platform/current/streams/concepts.html>

- Sugiyono, & Lestari, P. (2021). *Metode Penelitian Komunikasi(Kuantitatif, Kualitatif, dan Cara Mudah Menulis Artikel pada Jurnal Internasional)* [Digital]. Alfabeta. <http://eprints.upnyk.ac.id/27727/1/Buku%20Metode%20Penelitian%20Komunikasi.pdf>
- Tsai, W., Bai, X., Paul, R. C., Shao, W., & Agarwal, V. (2001). End-to-end integration testing design. <https://doi.org/10.1109/cmpsac.2001.960613>
- Umer, M., Mahesh, B., Hanson, L., M.R.Khabbazi, & Onori, M. (2018). Smart Power Tools : An Industrial Event-Driven Architecture Implementation. *51st CIRP Conference on Manufacturing Systems.* <https://doi.org/10.1016/j.procir.2018.03.058>
- Wang, H., & Sun, C. (2011). Game reward systems: Gaming experiences and social meanings. In *Digital Games Research Association Conference* (Vol. 6). <http://www.digra.org/wp-content/uploads/digital-library/11310.20247.pdf>
- What is a reasonable code coverage % for unit tests (and why)?* (n.d.). Stack Overflow. Retrieved February 10, 2023, from <https://stackoverflow.com/questions/90002/what-is-a-reasonable-code-coverage-for-unit-tests-and-why>
- What is Change Data Capture? How Does It Work?* (n.d.). Hazelcast. Retrieved June 8, 2023, from <https://hazelcast.com/glossary/change-data-capture/>
- Winberg, S., & Singh, S. (n.d.). Real-Time Event-driven Air Quality Inspection Framework for City-wide Pollution Level Monitoring. *Proc. of the 3rd International Conference on Electrical, Communication and Computer Engineering (ICECCE).* 978-1-6654-3897-1/21
- Windows.* Windows | Apache Flink. (n.d.). <https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/dev/datastream/operators/windows/>
- Winkowska, J., Szpilko, D., & Pejić, S. (2019). Smart city concept in the light of the literature review. *Engineering Management in Production and Services, 11(2),* 70–86. <https://doi.org/10.2478/emj-2019-0012>

Zhong, H., & Huang, L. (2016). The Empirical Research on the Consumers' Willingness to Participate in E-waste Recycling with a Points Reward System. *Energy Procedia*, 104, 475–480. <https://doi.org/10.1016/j.egypro.2016.12.080>

van Dongen, G., & Van den Poel, D. (2020). Evaluation of stream processing frameworks. *IEEE Transactions on Parallel and Distributed Systems*, 31(8), 1845–1858.

LAMPIRAN

Lampiran 1: Tabel Daftar Ambang Polutan untuk Konversi AQI

		Setelah itu						
...and this category	...equal this AQI	NO ₂ (ppb) 1-hour	SO ₂ (ppb) 1-hour	CO (ppm) 8-hour	PM ₁₀ (pg/m3) 24-hour	PM _{2.5} (pg/m3) 24-hour	O ₃ (ppm) 1-hour ¹	O ₃ (ppm) 8-hour
Good	0 - 50	0 - 53	0 - 35	0 - 4.4	0 - 54	0.0 - 12.0	-	0.000 - 0.054
Moderate	51 - 100	54 - 100	36 - 75	4.5 - 9.4	55 - 154	12.1 - 35.4	-	0.055 - 0.070
Unhealthy for	101 - 150	101 - 360	76 - 185	9.5 - 12.4	155 - 254	35.5 - 55.4	0.125 - 0.164	0.071 - 0.085
Unhealthy	151 - 200	361 - 649	(186 - 304) ⁴	12.5 - 15.4	255 - 354	(55.5 - 150.4) ³	0.165 - 0.204	0.086 - 0.105
Very unhealthy	201 - 300	650 - 1249	(305 - 604) ⁴	15.5 - 30.4	355 - 424	(150.5 - (250.4) ³	0.205 - 0.404	0.106 - 0.200
Hazardous	301 - 400	1250 - 1649	(605 - 804) ⁴	30.5 - 40.4	425 - 504	(250.5 - 350.4) ³	0.405 - 0.504	-2
Hazardous	401 - 500	1650 - 2049	(805 - 1004) ⁴	40.5 - 50.4	505 - 604	(350.5 - 500.4) ³	0.505 - 0.604	-2

Lampiran 2: Hasil Wawancara

Hari/Tanggal		Jumat/3 Maret 2023	
Nama Responden		Muhammad Falihadib	
Pewawancara		Natasya Zahra	
<i>User Segment</i>		Pengguna aplikasi lingkungan	
No	Topik	Pertanyaan	Jawaban
1	Latar Belakang Responden	Nama	Muhammad Falihadib
		Usia	21
		Pekerjaan	Mahasiswa
		Domisili	Jakarta
		Apakah Anda pernah menggunakan layanan atau aplikasi pemantau kualitas udara?	Iya
2	Pencemaran Udara	Apakah Anda tahu mengenai permasalahan pencemaran udara? Jika iya, jelaskan apa saja yang Anda ketahui mengenai itu. [Jika tidak, jelaskan permasalahan pencemaran udara]	Iya tahu, yang saya tahu itu jika melihat di media berita ada berita mengenai polusi udara atau udara yang tercemar entah untuk skala Indonesia atau Jakarta (Indonesia/Jakarta merupakan negara/kota dengan polusi udara ter-[kata sifat])
		Menurut Anda, apa penyebab dan bagaimana pencemaran udara itu terjadi? Apa hal-hal yang dilakukan sehari-hari yang dapat menyumbang polusi udara?	Ada banyak penyebabnya, kalau di Indonesia sebabnya seputar transportasi motor atau mobil, kemudian pembakaran sampah, asap rokok
		Menurut Anda, bagaimana kita bisa tahu bahwa lingkungan sekitar mengalami pencemaran atau polusi udara?	Memakai aplikasi pemantau kualitas udara, melihat berita, batuk-batuk
		Bagaimana kualitas udara di lingkungan sekitar Anda? Apa dan bagaimana dampak dari kualitas udara yang buruk ke sekitar?	Sedang-sedang saja tapi juga tidak terlalu baik
		Langkah apa yang pernah atau dapat Anda lakukan untuk setidaknya mencegah atau mengurangi pencemaran udara?	Naik transportasi umum, jalan, atau naik sepeda; hindari bakar sampah atau merokok

Lampiran 2: Hasil Wawancara (Lanjutan)

No	Topik	Pertanyaan	Jawaban
3	Layanan atau Aplikasi Pemantau Kualitas Udara (LAPKU)	Apa saja LAPKU yang pernah atau baru-baru ini digunakan?	Nafas, AQI Monitoring and weather forecast, Air Visual
		Apa <i>platform</i> yang Anda gunakan untuk mengakses LAPKU tersebut?	Android
		Seberapa sering intensitas Anda ketika menggunakan LAPKU dalam seminggu?	Jarang 0-1
		Apa tujuan utama atau alasan Anda pernah atau masih menggunakan LAPKU?	Untuk mengetahui informasi kualitas udara, dan penasaran
		Ketika Anda menggunakan LAPKU, adakah kemudahan yang Anda hadapi? Jelaskan	Tampilannya <i>simple compact</i> untuk Nafas, lengkap informasinya
		Ketika Anda menggunakan LAPKU, adakah rintangan yang Anda hadapi? Jelaskan. Jika ada, hal apa yang dapat mempermudah Anda menghadapi rintangan atau kesulitan yang dihadapi?	Pencarian yang ribet, tidak ada <i>map</i> , lalu daerahnya lingkup besar tidak per kecamatan misalnya. <i>overwhelm</i>
		Bagaimana Anda mendeskripsikan pengalaman dan perasaan Anda dalam menggunakan LAPKU?	Merasa menambah pengetahuan kayak "ohh", kalau Nafas cepat <i>detect</i> lokasi dan cukup cepat dapet informasi, kalo Air Visual lebih lengkap datanya dan membuat <i>amaze</i> , yang satu lagi banyak iklan dan mencarinya susah.

Lampiran 2: Hasil Wawancara (Lanjutan)

No	Topik	Pertanyaan	Jawaban
4	Fitur	Apa saja hal atau <i>task</i> yang Anda lakukan saat menggunakan LAPKU?	<i>Notif bar, search bar, map, memantau kualitas udara, prediksi, shop</i> (Nafas, Air Visual), artikel dan berita (Air Visual <i>worldwide</i>), statistik
	Setiap <i>Task</i> yang Disebutkan	[Jika menyebutkan banyak LAPKU] Apa perbedaan dalam melakukan [<i>task</i>] di setiap LAPKU?	Berbeda informasi artikel/ <i>news</i> dan <i>scope</i> daerah, ada yang <i>worldwide</i> dan indo; ada yang lengkap ada yang tidak; sumber dana aplikasi (<i>shop</i> atau iklan)
		Apa yang Anda lakukan untuk memenuhi [<i>task</i>] tersebut dan bagaimana?	Tinggal (melakukan) eksplorasi, ada <i>navbar</i> di bawah tinggal navigasi saja (Nafas dan Air Visual)
		Informasi apa yang Anda butuhkan atau ingin diperoleh dalam mencapai [<i>task</i>] tersebut?	AQI, PM2.5, suhu, CO2, kelembaban, lokasi, <i>maps</i>
		Apakah ada hal yang anda suka dalam melakukan [<i>task</i>]. Jika iya, jelaskan.	Visualnya tampilannya enak, contohnya <i>maps</i> infonya <i>obvious</i> warnanya, <i>highlight</i> poin-poin penting, <i>layout</i> bagus
		Apakah ada hal yang anda tidak suka dalam melakukan [<i>task</i>]. Jika iya, jelaskan.	Iklan dan <i>navigation</i> yang tidak ada (<i>one page</i>)
		Apakah Anda merasa kesulitan dalam melakukan [<i>task</i>]? Jika iya, hal apa yang sekiranya dapat mempermudah Anda dalam melakukan [<i>task</i>]? Jika tidak, hal apa yang membuat Anda merasa hal tersebut?	(Untuk orang) awam butuh informasi lebih lanjut tapi masih terbantu dengan tombol (i), navigasinya agak susah

Lampiran 2: Hasil Wawancara (Lanjutan)

No	Topik	Pertanyaan		Jawaban
4	Fitur	Setiap <i>Task</i> yang Disebutkan	Apakah ada masukan atau saran terkait [<i>task</i>] tersebut?	<i>Dark mode</i>
			Bagaimana pendapat Anda jika kami menyediakan [<i>task</i>] tersebut pada layanan Mahoni?	Oke
			Bagaimana pendapat Anda jika kami tidak menyediakan [<i>task</i>] tersebut pada layanan Mahoni?	Fitur statistik gak apa-apa gak ada, artikel gap apa-apa gak ada, yang penting mainnya (perannya) <i>location</i> ada
5	Penutup	Apakah ada masukan atau saran terkait pengembangan sistem Mahoni?		Ditemakan aplikasinya secara tampilan sesuai nama aplikasi
		Apakah ada pertanyaan yang saya belum tanyakan yang menurut Anda mungkin bermanfaat untuk kami ketahui?		
		Apakah ada sesuatu yang Anda ingin elaborasi atau tanyakan?		

Lampiran 2: Hasil Wawancara (Lanjutan)

Hari/Tanggal		Sabtu/4 Maret 2023	
Nama Responden		Deas Sativa Hasna Dinanti	
Pewawancara		Natasya Zahra	
<i>User Segment</i>		Pengguna sistem tukar poin menjadi kupon	
No	Topik	Pertanyaan	Jawaban
1	Latar Belakang Responden	Nama	Deas Sativa Hasna Dinanti
		Usia	21 Tahun
		Pekerjaan	Mahasiswa
		Domisili	Tangerang Selatan
		Apakah Anda pernah menggunakan sistem tukar poin menjadi kupon?	Iya
2	Sistem Tukar Poin Menjadi Kupon (STPMK)	Apa saja aplikasi atau layanan STPMK yang pernah atau baru-baru ini digunakan?	Gojek dan Telkomsel
		Apa <i>platform</i> yang Anda gunakan untuk mengakses STPMK tersebut?	<i>Handphone</i>
		Seberapa sering intensitas Anda ketika menggunakan STPMK dalam seminggu?	0-1
		Apa tujuan utama atau alasan Anda pernah atau masih menggunakan SPTMK?	Karena poin bisa digunakan untuk keperluan lain, poin gojek ditukar dengan Diskon Gofood 20k, sedangkan poin telkomsel ditukar dengan diskon voucher Zalora dan tambahan Gigabyte kuota paket internet
		Apa hal atau faktor yang sekiranya dapat memotivasi Anda menggunakan STPMK?	Karena mau mendapat diskon tertentu, takut jika poinnya hangus, sayang tidak dipakai
		Ketika Anda menggunakan SPTMK, adakah kemudahan yang Anda hadapi? Jelaskan	Ketika uang cashless tidak mencukupi, bisa menggunakan diskon hasil penukaran poin tersebut

Lampiran 2: Hasil Wawancara (Lanjutan)

No	Topik	Pertanyaan	Jawaban
2	Sistem Tukar Poin Menjadi Kupon (STPMK)	Ketika Anda menggunakan SPTMK, adakah rintangan yang Anda hadapi? Jelaskan. Jika ada, hal apa yang dapat mempermudah Anda menghadapi rintangan atau kesulitan yang dihadapi?	Terkadang harus menyelesaikan misi tertentu, seperti login harian dan membeli paket di Telkomsel atau memakai terus-menerus jasa dari aplikasi tersebut (seperti pakai gofood/goride dapat exp tertentu) dan dikumpulkan untuk ditukar nantinya
		Bagaimana Anda mendeskripsikan pengalaman dan perasaan Anda dalam menggunakan SPTMK?	Sejauh ini puas karena dapat meningkatkan kesetiaan dalam penggunaan juga karena kalo pakai terus jadi dapet poin yang bisa ditukar, tetapi jika ingin menukar ke diskon tertentu, biasanya membutuhkan banyak poin dan sulit untuk mengumpulkannya dalam waktu cepat
		Jenis atau kategori kupon apa yang menarik bagi Anda?	Diskon pakai goride/gofood atau cashback ketika membeli kuota internet
		Jenis atau kategori kupon apa yang pernah Anda beli?	Diskon pakai goride/gofood atau cashback ketika membeli kuota internet
3	Fitur	Apa saja hal atau <i>task</i> yang Anda lakukan saat menggunakan SPTMK?	Misi login harian di telkomsel dan mengumpulkan poin pembelian paket internet, memakai jasa gojek atau mengerjakan misi untuk mendapatkan exp tertentu yang dimana jika meraih tingkatan tertentu dapat menukar kepada voucher tertentu
		Setiap <i>Task</i> yang Disebutkan	[Jika menyebutkan banyak SPTMK] Apa perbedaan dalam melakukan [<i>task</i>] di setiap SPTMK?

Lampiran 2: Hasil Wawancara (Lanjutan)

No	Topik	Pertanyaan	Jawaban
3	Fitur	Setiap <i>Task</i> yang Disebutkan	<p>Apa yang Anda lakukan untuk memenuhi <i>[task]</i> tersebut dan bagaimana?</p> <ul style="list-style-type: none"> - Membuka aplikasi Telkomsel setiap hari untuk mengambil poin harian, membeli paket internet setiap bulannya - Membeli makanan melalui aplikasi gofood untuk mengumpulkan exp
			<p>Informasi apa yang Anda butuhkan atau ingin diperoleh dalam mencapai <i>[task]</i> tersebut?</p> <ul style="list-style-type: none"> - Sudah sejauh mana pencapaian misi yang didapat - Bagaimana memperoleh poin tersebut - Jumlah poin yang didapat - Syarat dan ketentuan voucher - Butuh berapa poin untuk memperoleh voucher tersebut - Kadaluarsa poin - Daftar voucher yang bisa didapat - Tingkatan poin dari pelanggan dan benefit dari memperoleh poin tertentu
			<p>Apakah ada hal yang anda suka dalam melakukan <i>[task]</i>. Jika iya, jelaskan.</p>
			<p>Apakah ada hal yang anda tidak suka dalam melakukan <i>[task]</i>. Jika iya, jelaskan.</p>
			<p>Apakah Anda merasa kesulitan dalam melakukan <i>[task]</i>? Jika iya, hal apa yang sekiranya dapat mempermudah Anda dalam melakukan <i>[task]</i>? Jika tidak, hal apa yang membuat Anda merasa hal tersebut?</p>

Lampiran 2: Hasil Wawancara (Lanjutan)

No	Topik	Pertanyaan		Jawaban
3	Fitur	Setiap <i>Task</i> yang Disebutkan	Apakah ada masukan atau saran terkait [<i>task</i>] tersebut?	-
			Bagaimana pendapat Anda jika kami menyediakan [<i>task</i>] tersebut pada layanan Mahoni?	Boleh
			Bagaimana pendapat Anda jika kami tidak menyediakan [<i>task</i>] tersebut pada layanan Mahoni?	Kalo mau pengguna setia memakai aplikasi, sebaiknya sistem poin kupon ini digunakan
			Skala kepentingan (1-5)	4
4	Penutup	Apakah ada masukan atau saran terkait pengembangan sistem Mahoni?		Partnership dengan aplikasi lain supaya vouchernya tidak terbatas pada penggunaan transportasi, tapi bisa makanan/minuman/wisata, dll
		Apakah ada pertanyaan yang saya belum tanyakan yang menurut Anda mungkin bermanfaat untuk kami ketahui?		-
		Apakah ada sesuatu yang Anda ingin elaborasi atau tanyakan?		-

Lampiran 3: Kuesioner Kano

User Research Fitur Mahoni

Responden yang terformat.

Perkenalkan kami Mohammad Riswanda Alifrahman, Muhammad Fathur Muthahhari dan Natasya Zahra merupakan mahasiswa tingkat akhir Fakultas Ilmu Komputer Universitas Indonesia sedang mengelakukan penelitian intik, proyek Tugas Akhir kami yang berjudul Mahoni.

Mahoni merupakan aplikasi Smart City yang mendukung penggunaanya untuk mempermudah transportasi umum dengan sistem reward. Reward yang diperoleh berupa poin yang berasal dari keadaan kualitas udara selama perjalanan menggunakan transportasi umum. Poin tersebut nantinya dapat ditukarkan menjadi berbagai macam kupon yang menarik.

Penjelasan singkat berupa video dapat dilihat pada tautan berikut:

<https://youtu.be/DjgnK2TXDmU>

Pendekatan ini bertujuan untuk melakukan validasi fitur-fitur serta mendapatkan masukan berupa saran yang nantinya akan memengaruhi implementasi sistem reward. Reward yang diperoleh berupa poin yang berasal dari kriteria responden yang saat ini berdomisili di Jabodetabek.

Pengisian kuesioner ini membutuhkan waktu 5-10 menit. Data yang dikumpulkan akan digunakan untuk kerjasamanya dan hanya akan digunakan untuk kebutuhan penelitian saja. Sebagai bentuk apresiasi, kami mengadakan undian dengan total sebesar Rp500.000 untuk 10 responden yang beruntung.

Atas waktu dan kesedian Anda untuk berpartisipasi dalam penelitian ini, kami ucapan terima kasih.

Salam,
Riswanda, Fathan, dan Natasya

Narahubung:
Natasya (natasya.zahra91@ui.ac.id)

* Indicates required question

1. Apakah Anda bersedia untuk berpartisipasi sebagai responden dalam penelitian ini? *

Mark only one oval.

- Ya, bersedia
- Tidak bersedia

Identitas Responden

Pada bagian ini, Anda akan mengisi informasi berupa data diri.

Peraturan pengisian :

Jawablah pertanyaan berikut dengan mengisi jawaban singkat atau memilih salah satu opsi jawaban yang tersedia.

6. Domisili *

Kota atau kabupaten

Mark only one oval.

- Jakarta
- Bogor
- Depok
- Tangerang
- Bekasi

7. Nomor WhatsApp atau ID Line yang dapat dihubungi

Untuk keperluan undian

Skip to question 8

Penggunaan Aplikasi Kualitas Udara

Pada bagian ini, Anda akan menjawab pertanyaan mengenai pengalaman menggunakan aplikasi atau layanan pemantau kualitas udara.

Peraturan pengisian :

Jawablah pertanyaan berikut dengan memilih salah satu opsi jawaban yang tersedia.

8. Apakah Anda pernah menggunakan aplikasi atau layanan pemantau kualitas udara? Contoh : * (Nafas, weathert.com)

Mark only one oval.

- Ya, pernah Skip to question 9
- Tidak pernah Skip to question 14

Skip to question 9

Penggunaan Aplikasi Kualitas Udara

Pada bagian ini, Anda akan menjawab pertanyaan mengenai pengalaman menggunakan aplikasi atau layanan pemantau kualitas udara.

Peraturan pengisian :

Jawablah pertanyaan berikut dengan mengisi jawaban singkat atau memilih salah satu opsi jawaban yang tersedia.

9. Aplikasi atau layanan kualitas udara apa yang pernah Anda gunakan? *

Sebutkan nama aplikasi atau layanan kualitas udara

2. Nama atau inisial *

3. Jenis kelamin *

Mark only one oval.

- Laki-laki
- Perempuan

4. Usia *

Mark only one oval.

- Di bawah 18 tahun
- 18 – 24 tahun
- 25 – 34 tahun
- 35 – 44 tahun
- Di atas 45 tahun

5. Pekerjaan *

Mark only one oval.

- Tidak belum bekerja
- Pelajar/Mahasiswa
- Ibu Rumah Tangga
- Pegawai Negeri Sipil
- Wirausahawan
- Pegawai Swasta
- Profesional (Dokter, Tenaga Pendidik, Konsultan, dsb)
- Other: _____

10. Apa tujuan Anda dalam menggunakan aplikasi atau layanan kualitas udara tersebut? *

Mark only one oval.

- ingin mengetahui informasi mengenai kualitas udara
- ingin menggunakan layanan lain yang ditawarkan aplikasi atau layanan kualitas udara
- Aplikasi atau layanan kualitas udara tersebut memiliki tampilan yang menarik
- Karena iklan atau promosi
- ingin mencoba saja atau penasaran
- Other: _____

11. Aspek dari aplikasi atau layanan kualitas udara tersebut apa yang Anda suka? *

Tick all that apply.

- Informasi mengenai kualitas udara yang disediakan lengkap
- Informasi mengenai kualitas udara yang disediakan akurat
- Terdapat fitur penunjang selain fitur informasi kualitas udara saat ini
- Tampilan aplikasi atau layanan yang menarik
- Other: _____

12. Aspek dari aplikasi atau layanan kualitas udara tersebut apa yang Anda tidak suka? *

Tick all that apply.

- Informasi mengenai kualitas udara yang disediakan tidak lengkap
- Informasi mengenai kualitas udara yang disediakan tidak akurat
- Tidak terdapat fitur penunjang selain fitur informasi kualitas udara saat ini
- Tampilan aplikasi atau layanan yang tidak menarik
- Other: _____

13. Informasi mengenai udara apa saja yang Anda butuhkan dalam aplikasi atau layanan kualitas udara? *

Tick all that apply.

- Air Quality Index (AQI)
- PM2.5
- Temperatur
- Kelembaban
- CO2
- Other: _____

Skip to question 15

Lampiran 3: Kuesioner Kano (Lanjutan)

Penggunaan Aplikasi Kualitas Udara

Pada bagian ini, Anda akan menjawab pertanyaan mengenai pengalaman menggunakan aplikasi atau layanan pemerintah kualitas udara.

Petunjuk pengisian :

Jawablah pertanyaan berikut dengan mengisi jawaban singkat atau memilih salah satu opsi jawaban yang tersedia.

14. Mengapa Anda tidak pernah menggunakan aplikasi atau layanan kualitas udara? *

Mark only one oval.

- Tidak adanya keperluan untuk mengetahui informasi mengenai kualitas udara
- Tidak mengetahui terdapat aplikasi atau layanan yang menyediakan informasi mengenai kualitas udara
- Tidak begitu memahami konsep dari kualitas udara
- Aplikasi atau layanan kualitas udara tidak memiliki tampilan yang menarik
- Tidak tertarik menggunakankannya
- Other: _____

Skip to question 15

Riset Fitur Kualitas Udara Mahoni

Pada bagian ini, Anda akan menjawab pertanyaan mengenai fitur-fitur yang akan terdapat dalam aplikasi Mahoni dan memberikan masukan berupa saran.

Rancangan High-Fidelity Prototype fitur ini dapat diakses pada Pigma berikut:

[High-Fidelity Prototype Kualitas Udara Mahoni](#)

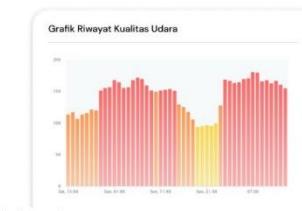
Petunjuk pengisian :

Jawablah pertanyaan berikut dengan mengisi jawaban singkat atau memilih salah satu opsi jawaban yang tersedia.

Keterangan:

- Suka = Suka atau senang apabila aplikasi ini ADA/TIDAK ADA suatu fitur
- Mengharapkan = Mengharapkan aplikasi ini ADA/TIDAK ADA suatu fitur karena kebutuhan
- Netral = Netral apabila aplikasi ini ADA/TIDAK ADA suatu fitur
- Dapat menerima = Dapat menerima apabila aplikasi ini ADA/TIDAK ADA suatu fitur
- Tidak suka = Tidak suka apabila aplikasi ini ADA/TIDAK ADA suatu fitur

15. Fitur riwayat kualitas udara di suatu tempat yang memuat informasi kualitas udara (Air Index Quality, temperatur, CO₂, dll) per satuan waktu, bagaimana perasaaan Anda ... *



Mark only one oval per row.

	Suka	Mengharapkan	Netral	Dapat menerima	Tidak suka
Jika ADA fitur ini?	<input type="radio"/>				
Jika TIDAK ADA fitur ini?	<input type="radio"/>				

16. Fitur pencarian yang memberikan informasi kualitas udara di tempat lain, bagaimana perasaaan Anda ... *



Mark only one oval per row.

	Suka	Mengharapkan	Netral	Dapat menerima	Tidak suka
Jika ADA fitur ini?	<input type="radio"/>				
Jika TIDAK ADA fitur ini?	<input type="radio"/>				

17. Fitur prediksi yang memberikan prediksi kualitas udara, bagaimana perasaaan Anda ... *



Mark only one oval per row.

	Suka	Mengharapkan	Netral	Dapat menerima	Tidak suka
Jika ADA fitur ini?	<input type="radio"/>				
Jika TIDAK ADA fitur ini?	<input type="radio"/>				

Lampiran 3: Kuesioner Kano (Lanjutan)

18. Fitur rekomendasi yang memberikan rekomendasi kegiatan berdasarkan kualitas udara saat ini, bagaimana perasaan Anda ... *

Rekomendasi Kesehatan

 Hindari aktivitas di luar  Nyalakan pemurni udara
 Tutup jendela anda untuk menghindari udara luar yang kotor  Kenakan masker di luar

Mark only one oval per row:

19. Fitur artikel mengenai kualitas udara untuk memperluas wawasan dan menambah pengetahuan, bagaimana perasaan Anda ... *

Artikel Apa itu AQI?




Apa itu AQI dan bagaimana cara menghitungnya?

Ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Mark only one oval per row:

20. Fitur *push-notification* yang dapat memberitahukan kualitas udara saat ini melalui ringkasnya notifikasi di *smartphone*, bagaimana perasaan Anda ... *

Mahoni 30m

Yuk beraktivitas di luar ruangan!

Skor Air Index Quality saat ini di tempatmu sebesar 50 (Baik). Kondisi saat ini cocok untuk beraktivitas ...

Mark only one oval per row:

21. Saran atau masukan terhadap fitur penting lainnya terkait dengan kualitas udara yang dapat ditambahkan Isi "-" jika tidak ada

Skip to question 22

Penggunaan Transportasi Umum

Pada bagian ini, Anda akan menjawab pertanyaan mengenai pengalaman menggunakan transportasi umum dalam kota.

Petunjuk pengisian :
Jawablah pertanyaan berikut dengan mengisi jawaban singkat atau memilih salah satu opsi jawaban yang tersedia.

Penggunaan Transportasi Umum

Pada bagian ini, Anda akan menjawab pertanyaan mengenai pengalaman menggunakan transportasi umum dalam kota.

Petunjuk pengisian :
Jawablah pertanyaan berikut dengan mengisi jawaban singkat atau memilih salah satu opsi jawaban yang tersedia.

23. Transportasi umum apa yang paling sering Anda gunakan? *

Mark only one oval.

- Kereta (KRL Commuter, MRT, LRT)
- Bis dalam kota (Transjakarta, dll)
- Angkot
- Other: _____

24. Berapa intensitas rata-rata Anda dalam penggunaan transportasi umum dalam seminggu? *

Mark only one oval.

- 0–2 kali
- 3–5 kali
- Lebih dari 5 kali

25. Berapa rata-rata waktu yang Anda butuhkan untuk menunggu transportasi umum tersebut * di tempat pemberhentian (stasiun, halte, dll)?

Mark only one oval.

- Tidak menunggu
- 1–5 menit
- 6–15 menit
- Lebih dari 15 menit

22. Apakah Anda pernah menggunakan transportasi umum dalam kota di seperti kereta atau bis?

Mark only one oval.

- Ya, pernah Skip to question 23
- Tidak pernah Skip to question 31

Lampiran 3: Kuesioner Kano (Lanjutan)

26. Mengapa Anda menggunakan transportasi umum ke suatu tempat? *

Tick all that apply.

- Dekat dengan tempat tujuan
- Harga murah
- Efisiensi waktu
- Tidak memiliki kendaraan pribadi
- Ingin mencoba saja
- Other: _____

27. Hal apa yang Anda sukai saat menggunakan transportasi umum? *

Tick all that apply.

- Tepat waktu
- Harga yang terjangkau
- Pelayanan yang baik
- Tempat menunggu atau keadaan kendaraan yang bersih
- Mesin kendaraan berjalan dengan baik
- Aman dari tindak kejahatan
- Other: _____

28. Hal apa yang Anda tidak sukai saat menggunakan transportasi umum? *

Tick all that apply.

- Kedatangan yang tidak menentu/terlambat
- Harga yang mahal
- Pelayanan yang buruk
- Tempat menunggu atau keadaan kendaraan yang kotor
- Mesin kendaraan tidak terawat
- Rawan dari tindak kejahatan
- Other: _____

29. Apakah Anda pernah menggunakan aplikasi layanan transportasi (aplikasi ojek online, aplikasi pendamping transportasi umum)? *

Mark only one oval.

- Ya, pernah
- Tidak pernah

30. Jelaskan mengapa Anda pernah/tidak pernah menggunakan aplikasi layanan transportasi *

Skip to question 32

Penggunaan Transportasi Umum

Pada bagian ini, Anda akan menjawab pertanyaan mengenai pengalaman menggunakan transportasi umum dalam kota.

Petunjuk pengisian :

Jawablah pertanyaan berikut dengan mengisi jawaban singkat atau memilih salah satu opsi jawaban yang tersedia.

31. Mengapa Anda tidak pernah menggunakan transportasi umum dalam kota? *

Tick all that apply.

- Tidak adanya keperluan untuk menggunakan transportasi umum
- Kedatangan kendaraan yang tidak menentu/terlambat
- Harga mahal
- Pelayanan yang buruk
- Tempat menunggu atau keadaan kendaraan yang kotor
- Rawan dari tindak kejahatan
- Tidak ingin mencobanya
- Other: _____

Skip to question 32

Riset Fitur Transportasi Mahoni

Pada bagian ini, Anda akan menjawab pertanyaan mengenai fitur-fitur yang akan terdapat dalam aplikasi Mahoni dan memberikan masukan berupa saran.

Rancangan High-Fidelity Prototype fitur ini dapat diakses pada Figma berikut:
[High-Fidelity Prototype Transportasi Mahoni](#)

Petunjuk pengisian :
Jawablah pertanyaan berikut dengan mengisi jawaban singkat atau memilih salah satu opsi jawaban yang tersedia.

Keterangan:

- Suka = Suka atau senang apabila aplikasi ini ADA/TIDAK ADA suatu fitur
- Mengharapkan = Mengharapkan aplikasi ini
- ADA/TIDAK ADA suatu fitur karena kebutuhan
- Neutral = Neutral apabila aplikasi ini
- ADA/TIDAK ADA suatu fitur
- Dapat menskoransi = Dapat menerima apabila aplikasi ini
- ADA/TIDAK ADA suatu fitur
- Tidak suka = Tidak suka apabila aplikasi ini
- ADA/TIDAK ADA suatu fitur

32. Fitur ringkas ini menyediakan informasi perjalanan Anda saat ini baik saat keberangkatan maupun saat tiba, bagaimana perasaan Anda ... *

Informasi Keberangkatan		Informasi Tiba	
Jenis Transportasi Waktu Lokasi Indeks Kualitas Ushra	09.00 Selasa, 21 April 2020 Terminal Monumen Nasional 89	Jln Lingga Waktu Lokasi Indeks Kualitas Ushra	09.00 Selasa, 21 April 2020 Taman Margonda Ragunan 43

Mark only one oval per row.

	Suka	Mengharapkan	Netral	Dapat menskoransi	Tidak suka
Jika ADA fitur ini?	<input type="radio"/>				
Jika TIDAK ADA fitur ini?	<input type="radio"/>				

Lampiran 3: Kuesioner Kano (Lanjutan)

33. Fitur riwayat yang menyediakan riwayat perjalanan Anda dalam menggunakan transportasi umum, bagaimana perasian Anda ... *

Mark only one oval per row.

	Suka	Mengharapkan	Netral	Dapat menoleransi	Tidak suka
Jika ADA fitur ini?	<input type="radio"/>				
Jika TIDAK ADA fitur ini?	<input type="radio"/>				

34. Saran atau masukan terhadap fitur penting lainnya terkait dengan transportasi umum yang * dapat ditambahkan

Isi "-" jika tidak ada

[Skip to question 35](#)

Penggunaan Sistem Poin dan Kupon

Pada bagian ini, Anda akan menjawab pertanyaan mengenai pengalaman menggunakan sistem poin dan kupon dalam suatu aplikasi. Pada sistem ini, pengguna aplikasi akan mendapatkan poin ketika telah menyelesaikan suatu task. Kemudian poin yang dikumpulkan nantinya dapat ditukarkan menjadi hadiah berupa kupon apabila jumlah poin tersebut telah memenuhi jumlah minimum untuk ditukarkan.

Petunjuk pengisian :

Jawablah pertanyaan berikut dengan memilih salah satu opsi jawaban yang tersedia.

Penggunaan Sistem Poin dan Kupon

Pada bagian ini, Anda akan menjawab pertanyaan mengenai pengalaman menggunakan sistem poin dan kupon dalam suatu aplikasi.

Petunjuk pengisian :

Jawablah pertanyaan berikut dengan mengisi jawaban singkat atau memilih salah satu opsi jawaban yang tersedia.

39. Mengapa Anda tidak pernah menggunakan sistem poin dan kupon? *

Mark only one oval.

- Tidak adanya keperluan untuk menggunakan sistem poin dan kupon
- Tidak mengetahui terdapat aplikasi atau layanan yang menyediakan sistem poin dan kupon
- Tidak begitu memahami konsep dari sistem poin dan kupon
- Tidak tertarik
- Other: _____

[Skip to question 40](#)

Riset Fitur Poin dan Kupon Mahoni

Pada bagian ini, Anda akan menjawab pertanyaan mengenai fitur-fitur yang akan terdapat dalam aplikasi Mahoni dan memberikan masukan berupa saran.

Rancangan High-Fidelity Prototype fitur ini dapat diakses pada Figma berikut:
[High-Fidelity Prototype Poin dan Kupon Mahoni](#)

Petunjuk pengisian :

Jawablah pertanyaan berikut dengan mengisi jawaban singkat atau memilih salah satu opsi jawaban yang tersedia.

Keterangan:

- * Suka = Suka atau senang apabila aplikasi ini ADA/TIDAK ADA suatu fitur
- * Mengharapkan = Mengharapkan aplikasi ini ADA/TIDAK ADA suatu fitur karena kebutuhan
- * Netral = Netral apabila aplikasi ini ADA/TIDAK ADA suatu fitur
- * Dapat menoleransi = Dapat menoleransi apabila aplikasi ini ADA/TIDAK ADA suatu fitur
- * Tidak suka = Tidak suka apabila aplikasi ini ADA/TIDAK ADA suatu fitur

40. Apakah Anda tertarik menggunakan transportasi umum jika mendapatkan poin yang dapat * ditukarkan menjadi sebuah kupon?

Mark only one oval.

- Ya, tertarik
- Tidak tertarik

35. Apakah Anda pernah menggunakan sistem tukar poin menjadi kupon dalam suatu aplikasi? *

Contoh aplikasi: MyTelkomsel, Tida

Mark only one oval.

- Ya, pernah [Skip to question 36](#)
- Tidak pernah [Skip to question 39](#)

Penggunaan Sistem Poin dan Kupon

Pada bagian ini, Anda akan menjawab pertanyaan mengenai pengalaman menggunakan sistem poin dan kupon dalam suatu aplikasi.

Petunjuk pengisian :

Jawablah pertanyaan berikut dengan mengisi jawaban singkat atau memilih salah satu opsi jawaban yang tersedia.

36. Aplikasi dengan sistem tukar poin apa yang pernah Anda gunakan? *

37. Apa alasan Anda menggunakan sistem tersebut? *

Mark only one oval.

- Mendapatkan promo dari suatu layanan/produk
- Mendapatkan keuntungan khusus sebagai pengguna
- Sayang jika poin tidak digunakan dan menjadi挂ang
- Gaya hidup hemat
- Ingin mencoba saja
- Other: _____

38. Faktor apa yang membuat Anda termotivasi untuk menggunakan sistem tukar poin? *

Tick all that apply.

- Merchant penyedia dan jenis kupon yang beragam
- Penawaran atau promo kupon yang menarik
- Pemaksaan poin menjadi kupon dengan nominal rendah
- Other: _____

[Skip to question 40](#)

41. Menurut Anda, kategori kupon apa yang menarik? *

Tick all that apply.

- Makanan dan minuman
- Pariwisata
- Telekomunikasi (pulsa, paket internet, e-money)
- Kesehatan dan kecantikan
- Hiburan (permainan, musik)
- Other: _____

42. Fitur ringkas yang menampilkan informasi jumlah poin yang dimiliki, bagaimana perasian Anda ... *



Mark only one oval per row.

	Suka	Mengharapkan	Netral	Dapat menoleransi	Tidak suka
Jika ADA fitur ini?	<input type="radio"/>				
Jika TIDAK ADA fitur ini?	<input type="radio"/>				

	Suka	Mengharapkan	Netral	Dapat menoleransi	Tidak suka
Jika ADA fitur ini?	<input type="radio"/>				
Jika TIDAK ADA fitur ini?	<input type="radio"/>				

Lampiran 3: Kuesioner Kano (Lanjutan)

43. Fitur riwayat yang menampilkan riwayat penkurbanan poin, bagaimana perasaan Anda ... *

Riwayat Redeem

Selasa, 21 April 2020

Kupon	Kupon Belanja Minimum	Poin
MICHAEL	Kupon Belanja Minimum 1	50 Poin
202001	Kupon Belanja Minimum 1	50 Poin
GIRONALU	Kupon Belanja Minimum 2	100 Poin

Semu, 20 April 2020

Mark only one oval per row.

	Suka	Mengharapkan	Netral	Dapat menoleransi	Tidak suka
Jika ADA fitur ini?	<input type="radio"/>				
Jika TIDAK ADA fitur ini?	<input type="radio"/>				

44. Fitur yang menampilkan daftar dan detail informasi kupon yang tersedia, bagaimana perasaan Anda ... *



Mark only one oval per row.

	Suka	Mengharapkan	Netral	Dapat menoleransi	Tidak suka
Jika ADA fitur ini?	<input type="radio"/>				
Jika TIDAK ADA fitur ini?	<input type="radio"/>				

45. Fitur yang menampilkan daftar dan detail informasi kupon yang dimiliki, bagaimana perasaan Anda ... *

Reward Kode

Kode kupon

1234 9012 8301 1284

Diskon Kupon Rp5.000 Dimsum

Tunjukkan transaksi di atau kepada waria Chinese Food
Dapatkan potongan belanja di Chinese Food sebesar Rp5.000.
Kupon dapat digunakan tanpa minimum pembelian:
Barang yang dapat ditukarkan salah sebagian berikut:
• Dimsum isi Ayam Biasa
• Dimsum isi Ayam Special

Mark only one oval per row.

	Suka	Mengharapkan	Netral	Dapat menoleransi	Tidak suka
Jika ADA fitur ini?	<input type="radio"/>				
Jika TIDAK ADA fitur ini?	<input type="radio"/>				

Terima kasih

Terima kasih atas partisipasinya! 🎉
Partisipasi Anda sangat berarti bagi kami. Semoga sehat dan bahagia selalu. Jangan lupa klik "Kirim" di bawah ini untuk mengumpulkan jawaban kuesioner.

Salam,
Riswanda, Fathan, dan Natasya

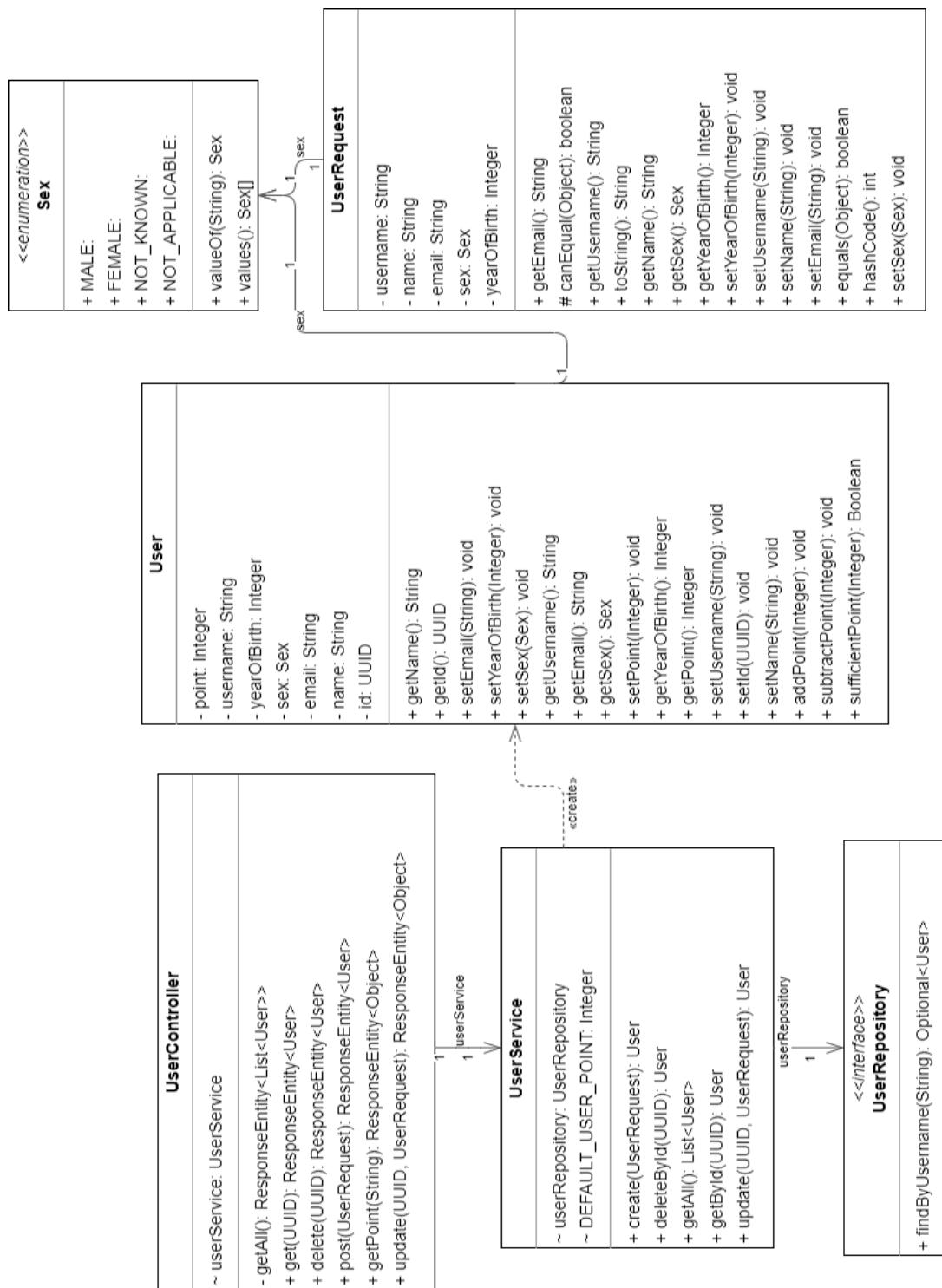
This content is neither created nor endorsed by Google.

Google Forms

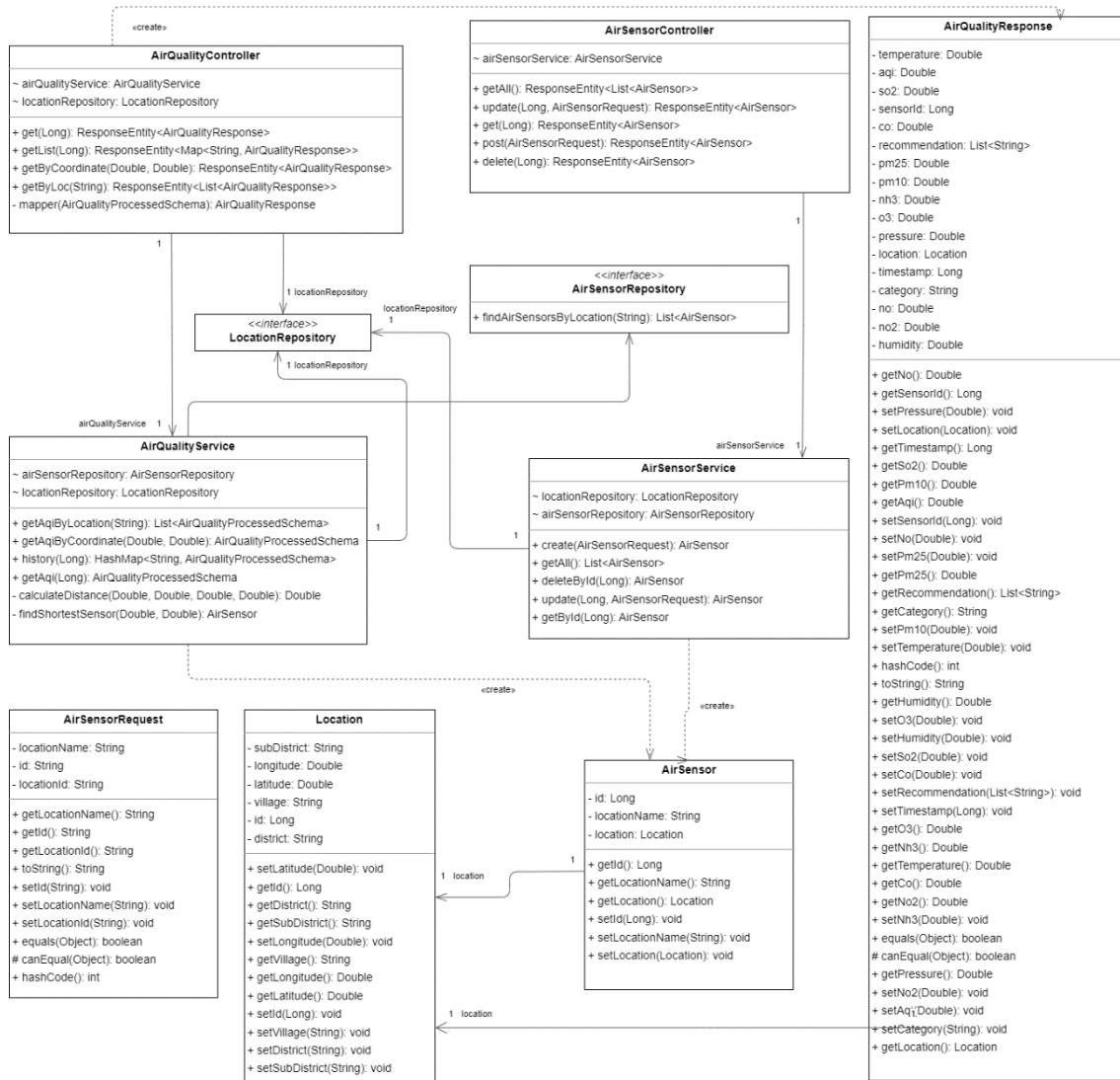
46. Saran atau masukan terhadap fitur penting lainnya terkait dengan sistem poin dan kupon yang dapat ditambahkan
Isi "-" jika tidak ada

Skip to section 15 (Terima kasih)

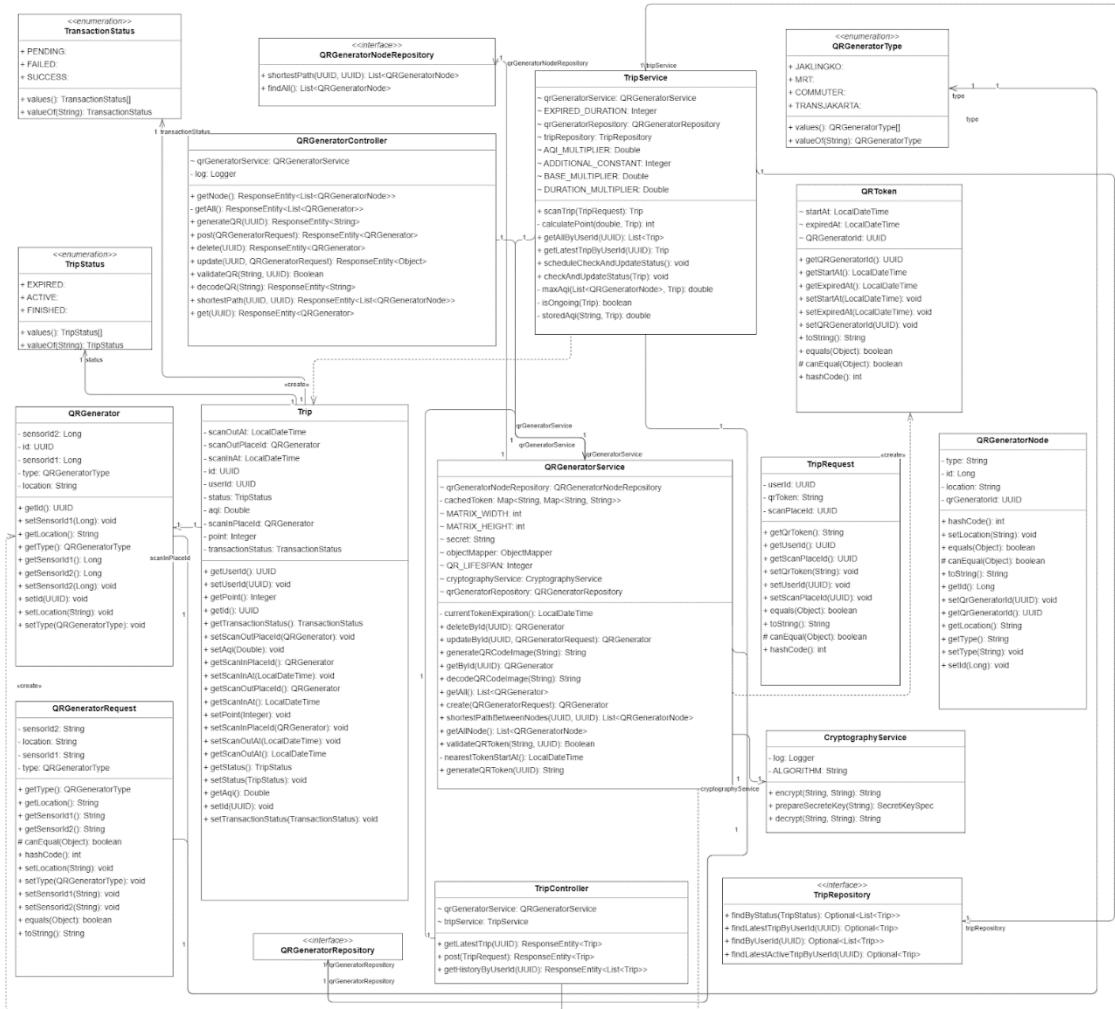
Lampiran 4: Class Diagram Servis Pengguna



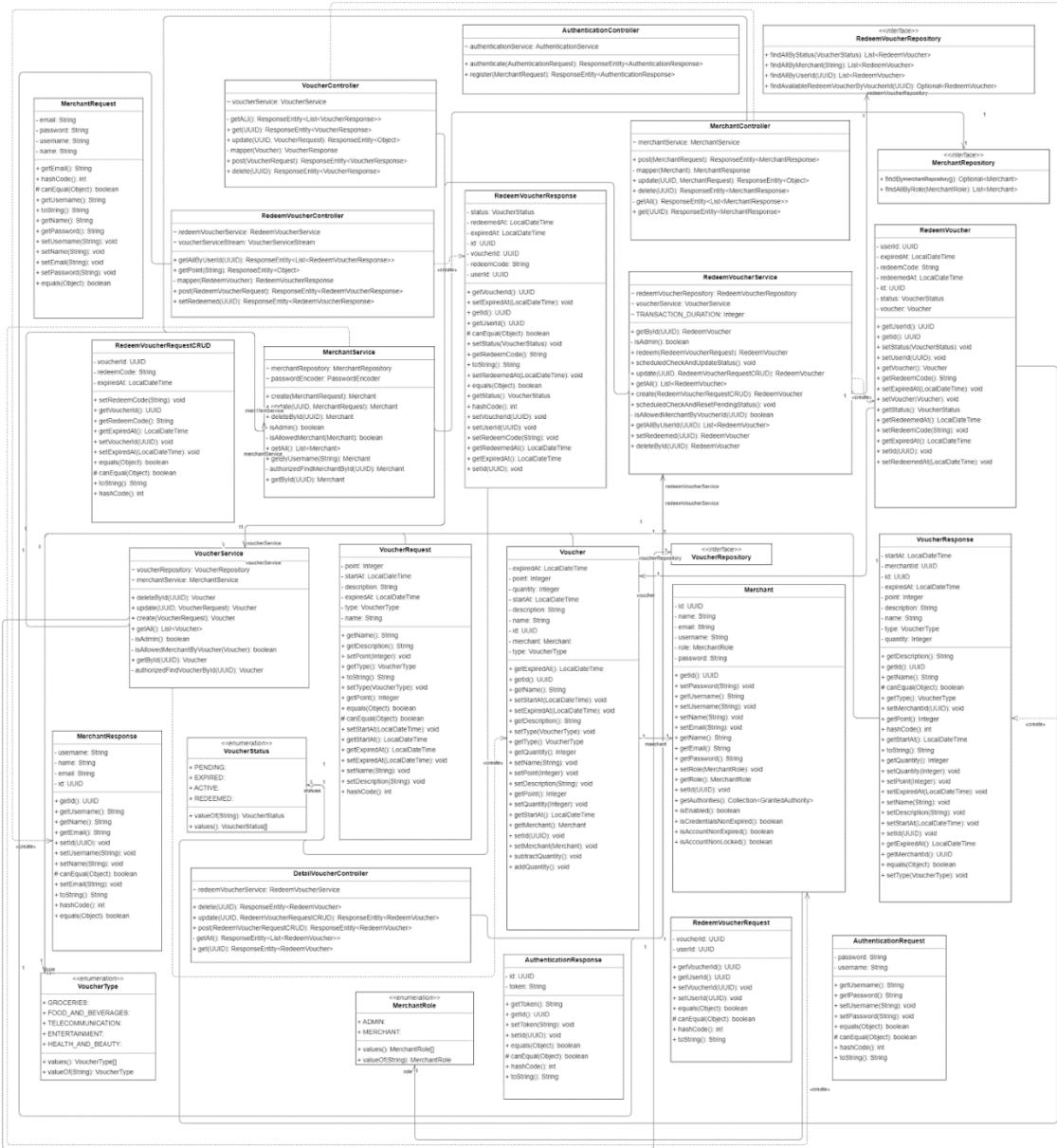
Lampiran 5: Class Diagram Servis Kualitas Udara



Lampiran 6: Class Diagram Servis Perjalanan



Lampiran 7: Class Diagram Servis Kupon dan Mitra Usaha



Lampiran 8: Daftar Seluruh Endpoint Mahoni

ID	URL	Method	Status	Name	Success	Role
API1	USER_HOST/api/v1/ users	GET	200	Get All User	✓	-
API2	USER_HOST/api/v1/ users	POST	200	Create User	✓	-
			400	Create User Bad Request	✓	-
API3	USER_HOST/api/v1/ users/{id}	GET	200	Get By Id	✓	-
			404	Get By Id Not Found	✓	-
API4	USER_HOST/api/v1/ users/{id}	DELETE	200	Delete User	✓	-
			404	Delete User Not Found	✓	-
API5	USER_HOST/api/v1/ users/{id}	PUT	200	Update User	✓	-
			404	Update User Not Found	✓	-
API6	AIR_HOST/api/v1/ air-sensors	GET	200	Get All Air Sensors	✓	-
API7	AIR_HOST/api/v1/ air-sensors	POST	200	Create Air Sensor	✓	-
			400	Create Air Sensor Bad Request	✓	-
API8	AIR_HOST/api/v1/ air-sensors/{id}	GET	200	Get By Id	✓	-
			404	Get By Id Not Found	✓	-
API9	AIR_HOST/api/v1/ air-sensors/{id}	DELETE	200	Delete Air Sensor	✓	-
			404	Delete Air Sensor Not Found	✓	-
API10	AIR_HOST/api/v1/ air-sensors/{id}	PUT	200	Update Air Sensor	✓	-
			404	Update Air Sensor Not Found	✓	-
API11	AIR_HOST/api/v1/ air-quality/id/{id}	GET	200	Get Air Quality	-	
			404	Get Air Quality Not Found	✓	-

Lampiran 8: Daftar Seluruh Endpoint Mahoni (Lanjutan)

ID	URL	Method	Status	Name	Success	Role
API12	AIR_HOST/api/v1/ air-quality/loc/{loc}	GET	200	Get Air Quality By Location	✓	-
			404	Get Air Quality By Location Not Found	✓	-
API13	AIR_HOST/api/v1/ air-quality/coordinate ?longitude={long}&l atitude={lat}	GET	200	Get Air Quality By Coordinate	✓	-
API 14	AIR_HOST/api/v1/ air-quality/history/ {id}	GET	200	Get History	✓	-
API15	TRIP_HOST/api/v1/ qr-generators	GET	200	Get All QR Generator	✓	-
API16	TRIP_HOST/api/v1/ qr-generators	POST	200	Add QR Generator	✓	-
			400	Add QR Generator Bad Request	✓	-
API17	TRIP_HOST/api/v1/ qr-generators/{id}	GET	200	Get By Id	✓	-
			404	Get By Id Not Found	✓	-
API18	TRIP_HOST/api/v1/ qr-generators/{id}	DELETE	200	Delete QR Generator	✓	-
			404	Delete QR Generator Not Found	✓	-
API19	TRIP_HOST/api/v1/ qr-generators/{id}	PUT	200	Update QR Generator	✓	-
			404	Update QR Generator Not Found	✓	-
API20	TRIP_HOST/api/v1/ qr-generators/{id}/ generate-qr	GET	200	Generate QR	✓	-

Lampiran 8: Daftar Seluruh Endpoint Mahoni (Lanjutan)

ID	URL	Method	Status	Name	Success	Role
API21	TRIP_HOST/api/v1/ trips	POST	200	Scan Trip	✓	-
API22	TRIP_HOST/api/v1/ trips/history/{userId}	GET	200	Get History Trip	✓	-
API23	TRIP_HOST/api/v1/ trips/latest-trip/ {userId}	GET	200	Get Latest Trip	✓	-
API24	VOUCHER_HOST/ api/v1/auth/register	POST	200	Register Merchant	✓	Public
API25	VOUCHER_HOST/ api/v1/auth/ authenticated	POST	200	Authenticate Merchant	✓	Public
API26	VOUCHER_HOST/ api/v1/merchants	GET	200	Get All Merchant	✓	Public
API27	VOUCHER_HOST/ api/v1/merchants	POST	200	Add Merchant	✓	Merchant, Admin
			404	Add Merchant	✓	Merchant, Bad Request
			403	Add Merchant	✓	Public Forbidden
API28	VOUCHER_HOST/ api/v1/merchants/ {id}	GET	200	Get By Id	✓	Public
			404	Get By Id Not Found	✓	Public
API29	VOUCHER_HOST/ api/v1/merchants/ {id}	DELETE	200	Delete Merchant	✓	Merchant, Admin
			403	Delete Merchant	✓	Public Not Logged In
			403	Delete Merchant	✓	Merchant Unauthorized
API30	VOUCHER_HOST/ api/v1/merchants/ {id}	PUT	200	Update Merchant	✓	Merchant, Admin
			403	Update Merchant	✓	Public Not logged In
			403	Update Merchant	✓	Public Unauthorized

Lampiran 8: Daftar Seluruh Endpoint Mahoni (Lanjutan)

ID	URL	Method	Status	Name	Success	Role
API31	VOUCHER_HOST/ api/v1/vouchers	GET	200	Get All Voucher	✓	Public
API32	VOUCHER_HOST/ api/v1/vouchers	POST	200	Add Voucher	✓	Merchant, Admin
			403	Add Voucher	✓	Public, Forbidden Merchant
API33	VOUCHER_HOST/ api/v1/vouchers/{id}	GET	200	Get By Id	✓	Public
			404	Get By Id Not Found	✓	Public
API34	VOUCHER_HOST/ api/v1/vouchers/{id}	DELETE	200	Delete Voucher	✓	Merchant, Admin
			403	Delete Voucher Not Logged In	✓	Public
			403	Delete Voucher Unauthorized	✓	Merchant
			404	Delete Voucher Not Found	✓	Merchant, Admin
API35	VOUCHER_HOST/ api/v1/vouchers/{id}	PUT	200	Update Voucher	✓	Merchant, Admin
			403	Update Voucher Not Logged In	✓	Public
			403	Update Voucher Unauthorized	✓	Merchant
			404	Update Voucher Not Found	✓	Merchant, Admin
API36	VOUCHER_HOST/ api/v1/vouchers/ detail	GET	200	Get All Redeem Voucher	✓	Merchant, Admin
			403	Get All Redeem Voucher Not Logged In	✓	Public

Lampiran 8: Daftar Seluruh Endpoint Mahoni (Lanjutan)

ID	URL	Method	Status	Name	Success	Role
API37	VOUCHER_HOST/ api/v1/vouchers/ detail	POST	200	Add Redeem Voucher	✓	Merchant, Admin
			403	Add Redeem Voucher Not Logged In	✓	Public
			400	Add Redeem Voucher Unauthorized	✓	Merchant
API38	VOUCHER_HOST/ api/v1/vouchers/ detail/{id}	GET	200	Get By Id	✓	Public
			403	Get By Id Not Logged In	✓	Merchant
			403	Get By Id Unauthorized	✓	Merchant, Admin
			404	Get By Id Not Found	✓	
API39	VOUCHER_HOST/ api/v1/vouchers/ detail/{id}	DELETE	200	Delete Redeem Voucher	✓	Merchant, Admin
			403	Delete Redeem Voucher Not Logged In	✓	Public
			403	Delete Redeem Voucher Unauthorized	✓	Merchant
			404	Delete Redeem Voucher Not Found	✓	Merchant, Admin

Lampiran 8: Daftar Seluruh Endpoint Mahoni (Lanjutan)

ID	URL	Method	Status	Name	Success	Role
API40	VOUCHER_HOST/ api/v1/vouchers/ detail/{id}	PUT	200	Update Redeem Voucher	✓	Merchant, Admin
			403	Update Redeem Voucher Not Logged In	✓	Public
			403	Update Redeem Voucher Unauthorized	✓	Merchant
			404	Update Redeem Voucher Not Found	✓	Merchant, Admin
API41	VOUCHER_HOST/ api/v1/redeem	POST	200	Redeem Voucher	✓	Public
API42	VOUCHER_HOST/ api/v1/redeem/{id}	POST	200	Set Redeemed	✓	Public
			404	Set Redeemed	✓	Public
API43	VOUCHER_HOST/ api/v1/redeem/user/ {id}	GET	200	Get All Redeem Voucher by User	✓	Public

Lampiran 9: Schema Domain Events Mahoni

Air Quality Domain		
AirQualityRawSchema	AirQualityProcessedSchema	AirQualityTable
<p>Schema Name: air-quality-raw-v1 Version: 1 Topic: air-quality-raw</p> <pre>{ "co": "double", "eventid": "string", "humidity": "double", "nh3": "double", "no": "double", "no2": "double", "o3": "double", "pm1": "double", "pm10": "double", "pm25": "double", "pressure": "double", "sensorid": "string", "so2": "double", "temperature": "double", "timestamp": "long" }</pre>	<p>Schema Name: air-quality-processed-v1 Version: 1 Topic: air-quality-processed</p> <pre>{ "category": "string", "co": "double", "district": "string", "eventid": "string", "humidity": "double", "idLocation": "long", "nameLocation": "string", "nh3": "double", "no": "double", "no2": "double", "o3": "double", "pm1": "double", "pm10": "double", "pm25": "double", "pressure": "double", "sensorid": "long", "so2": "double", "subdistrict": "string", "temperature": "double", "timestamp": "long" }</pre>	<p>Schema Name: air-quality-compacted-v1 Version: 1 Topic: air-quality-compacted</p> <p>Key: "day:hour:sensorid" Value = AirQualityProcessedSchema</p>

Merchant & Voucher Domain	Trip Domain
<p>VoucherRedeemedSchema</p> <p>Schema Name: voucher-redeemed-event-v1 Version: 1 Topic: voucher-redeemed-topic</p> <pre>{ "eventId": "string", "timestamp": "long", "voucherId": "string" "userId": "string" "code": "string" "point": "int", "redeemedAt": "long", "expiredAt": "long" }</pre>	<p>TripSchema</p> <p>Schema Name: trip-event-v1 Version: 1 Topic: trip-topic</p> <pre>{ "eventId": "string", "timestamp": "long", "tripId": "string" "userId": "string" "scanInPlaceId": "string" "scanInTimestamp": "long", "scanOutPlaceId": "string", "scanOutTimestamp": "long", "status": "string" (ACTIVE, EXPIRED, FINISHED) "aqi": "double", "point": "int" }</pre>

Lampiran 9: Schema *Domain Events* Mahoni (Lanjutan)

User Domain	
UserPointSchema	UserPointSchemaTable
Schema Name: user-point-v1 Version: 1 Topic: user-point-topic	Schema Name: user-point-table-v1 Version: 1 Topic: user-point-compacted-topic
{ "eventId": "string", "timestamp": "long", "userId": "string", "prevPoint": "int", "point": "int", "lastModifiedBy": string }	Key = "userId" Value = { "userId": "string" "point": "int" }

Lampiran 10: Response Body Hasil Pengujian End-to-End Testing Evaluasi Umum

```

MERCHANT
{
    "id": "a46602ff-0936-45a1-998b-23dd61395cad",
    "username": "Greyson.Windler",
    "name": "Greyson.Windler",
    "email": "Greyson.Windler@mail.com"
}

VOUCHER
{
    "id": "13e198db-5bb8-40b5-a442-023a965bee88",
    "name": "Awesome Wooden Gloves",
    "description": "Refined",
    "type": "TELECOMMUNICATION",
    "point": 5,
    "startAt": "2023-06-13T04:40:59.416",
    "expiredAt": "2025-01-01T00:00:00",
    "merchantId": "a46602ff-0936-45a1-998b-23dd61395cad"
    "quantity": 0
}

REDEEM VOUCHER
{
    "id": "29ca4ae0-a972-466e-8971-ac0d7ee78e36",
    "voucherId": "13e198db-5bb8-40b5-a442-023a965bee88",
    "userId": "67eb5009-fe79-4145-839c-bde98e9afa43",
    "redeemCode": "5zmDkqt9I_XufJE",
    "status": "REDEEMED",
    "redeemedAt": "2023-06-13T04:41:01.968844",
    "expiredAt": "2025-01-01T00:00:00"
}

USER
{
    "id": "67eb5009-fe79-4145-839c-bde98e9afa43",
    "username": "Kareem_Shields7",
    "name": "Kareem_Shields7",
    "email": "Kareem_Shields7@mail.com",
    "sex": "FEMALE",
    "yearOfBirth": 2000,
    "point": 65
}

```

**Lampiran 10: Response Body Hasil Pengujian End-to-End Testing Evaluasi Umum
(Lanjutan)**

```
TRIP
{
    "id": "d78d7079-6590-47b0-a97b-cf8d546a97eb",
    "userId": "67eb5009-fe79-4145-839c-bde98e9afa43",
    "scanInPlaceId": {
        "id": "5fedd2f9-9807-45fa-aab6-8efd74710400",
        "location": "Juanda",
        "type": "COMMUTER",
        "sensorId1": 87300,
        "sensorId2": 96571
    },
    "scanOutPlaceId": {
        "id": "8c974303-1ec4-47ba-a17c-ef1ac22933ae",
        "location": "Depok Baru",
        "type": "COMMUTER",
        "sensorId1": 80692,
        "sensorId2": 35824
    },
    "scanInAt": [
        2023,
        6,
        13,
        4,
        41,
        0,
        476064000
    ],
    "scanOutAt": [
        2023,
        6,
        13,
        4,
        41,
        0,
        815390000
    ],
    "status": "FINISHED",
    "aqi": 30.0,
    "point": 70,
    "transactionStatus": "SUCCESS"
}
```

Lampiran 11: Ringkasan Hasil Pengujian *End-to-End Testing Evaluasi Umum*

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Postman CLI	Performance Test	1	3s 814ms	18	43 ms
POST Register Merchant http://35.219.27.149:8080/api/v1/auth/register				200 OK 162 ms 200 B	
PASS Status code is 200					
POST Authenticate Merchant http://35.219.27.149:8080/api/v1/auth/authenticate				200 OK 107 ms 200 B	
PASS Status code is 200					
POST Add Voucher http://35.219.27.149:8080/api/v1/vouchers				200 OK 21 ms 272 B	
PASS Status code is 200					
POST Add Redeem Voucher http://35.219.27.149:8080/api/v1/vouchers/detail				200 OK 23 ms 759 B	
PASS Status code is 200					
POST Create User http://34.101.221.17:8080/api/v1/users				200 OK 51 ms 178 B	
PASS Status code is 200					
PASS Return Correct Data					
GET Generate QR In http://35.219.13.108:8080/api/v1/qr-generators/5feddf2f9-9807-45fa-aab6-8efd74710400/generate-qr				200 OK 25 ms 912 B	
PASS Status code is 200					
GET Scan QR In http://35.219.13.108:8080/api/v1/qr-generators/decode-qr?data=iVBORw0KGgoAAAANSUhEUgAAAMgAAADIAQAAAACFI5MzAAACc0IEQVR4Xu2WQY7klAxFHb...				200 OK 14 ms 152 B	
PASS Status code is 200					
POST Add Trip In http://35.219.13.108:8080/api/v1/trips				200 OK 26 ms 375 B	
PASS Status code is 200					
PASS Return Correct Data					
GET Generate QR Out http://35.219.13.108:8080/api/v1/qr-generators/8c974303-1ec4-47ba-a17c-ef1ac22933ae/generate-qr				200 OK 18 ms 936 B	
PASS Status code is 200					
GET Scan QR Out http://35.219.13.108:8080/api/v1/qr-generators/decode-qr?data=iVBORw0KGgoAAAANSUhEUgAAAMgAAADIAQAAAACFI5MzAAACg0IEQVR4Xu2XO86rMBCFB...				200 OK 15 ms 152 B	
PASS Status code is 200					
POST Scan Out Trip http://35.219.13.108:8080/api/v1/trips				200 OK 47 ms 523 B	
PASS Status code is 200					
PASS Return Correct Data					
POST Redeem Voucher http://35.219.27.149:8080/api/v1/redeem				200 OK 25 ms 258 B	
PASS Status code is 200					
PASS Return Correct Data					
POST Set Redeemed http://35.219.27.149:8080/api/v1/redeem/29ca4ae0-a972-466e-8971-ac0d7ee78e36				200 OK 19 ms 271 B	
PASS Status code is 200					
PASS Return Correct Data					

Lampiran 12 : File *deployment* Flink di Kubernetes Engine

```
flink-configuration-configmap.yaml

apiVersion: v1
kind: ConfigMap
metadata:
  name: flink-config
  labels:
    app: flink
data:
  flink-conf.yaml: |+
    jobmanager.rpc.address: flink-jobmanager
    taskmanager.numberOfTaskSlots: 2
    blob.server.port: 6124
    jobmanager.rpc.port: 6123
    taskmanager.rpc.port: 6122
    queryable-state.proxy.ports: 6125
    jobmanager.memory.process.size: 9000m
    taskmanager.memory.process.size: 10000m
    parallelism.default: 2
  log4j-console.properties: |+
    # This affects Logging for both user code and Flink
    rootLogger.level = INFO
    rootLogger.appenderRef.console.ref = ConsoleAppender
    rootLogger.appenderRef.rolling.ref = RollingFileAppender

    # Uncomment this if you want to _only_ change Flink's logging
    #logger.flink.name = org.apache.flink
    #logger.flink.level = INFO

    # The following lines keep the log level of common libraries/connectors on
    # Log Level INFO. The root logger does not override this. You have to manually
    # change the Log Levels here.
    logger.akka.name = akka
    logger.akka.level = INFO
    logger.kafka.name= org.apache.kafka
    logger.kafka.level = INFO
    logger.hadoop.name = org.apache.hadoop
    logger.hadoop.level = INFO
    logger.zookeeper.name = org.apache.zookeeper
    logger.zookeeper.level = INFO

    # Log all infos to the console
    appender.console.name = ConsoleAppender
    appender.console.type = CONSOLE
    appender.console.layout.type = PatternLayout
    appender.console.layout.pattern = %d{yyyy-MM-dd HH:mm:ss,SSS} %-5p %-60c %x -
    %m%n

    # Log all infos in the given rolling file
    appender.rolling.name = RollingFileAppender
    appender.rolling.type = RollingFile
    appender.rolling.append = false
    appender.rolling.fileName = ${sys:log.file}
```

Lampiran 12 : File deployment Flink di Kubernetes Engine (Lanjutan)

```
...
    appender.rolling.filePattern = ${sys:log.file}.%i
    appender.rolling.layout.type = PatternLayout
    appender.rolling.layout.pattern = %d{yyyy-MM-dd HH:mm:ss,SSS} %-5p %-60c %x -
%m%n
    appender.rolling.policies.type = Policies
    appender.rolling.policies.size.type = SizeBasedTriggeringPolicy
    appender.rolling.policies.size.size=100MB
    appender.rolling.strategy.type = DefaultRolloverStrategy
    appender.rolling.strategy.max = 10

    # Suppress the irrelevant (wrong) warnings from the Netty channel handler
    logger.netty.name = org.jboss.netty.channel.DefaultChannelPipeline
    logger.netty.level = OFF
```

```
jobmanager-service.yaml
apiVersion: v1
kind: Service
metadata:
  name: flink-jobmanager
spec:
  type: ClusterIP
  ports:
    - name: rpc
      port: 6123
    - name: blob-server
      port: 6124
  selector:
    app: flink
    component: jobmanager
```

```
jobmanager-service-ui.yaml
apiVersion: v1
kind: Service
metadata:
  name: flink-jobmanager
spec:
  type: LoadBalancer
  ports:
    - name: webui
      port: 8081
  selector:
    app: flink
    component: jobmanager
```

Lampiran 12 : File deployment Flink di Kubernetes Engine (Lanjutan)

```
jobmanager-session-deployment-non-ha.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: flink-jobmanager
spec:
  replicas: 1
  selector:
    matchLabels:
      app: flink
      component: jobmanager
  template:
    metadata:
      labels:
        app: flink
        component: jobmanager
    spec:
      containers:
        - name: jobmanager
          image: apache/flink:1.17.1-scala_2.12
          args: ["jobmanager"]
          ports:
            - containerPort: 6123
              name: rpc
            - containerPort: 6124
              name: blob-server
            - containerPort: 8081
              name: webui
          livenessProbe:
            tcpSocket:
              port: 6123
            initialDelaySeconds: 30
            periodSeconds: 60
          volumeMounts:
            - name: flink-config-volume
              mountPath: /opt/flink/conf
          securityContext:
            runAsUser: 9999 # refers to user _flink_ from official flink image,
change if necessary
      volumes:
        - name: flink-config-volume
      configMap:
        name: flink-config
        items:
          - key: flink-conf.yaml
            path: flink-conf.yaml
          - key: log4j-console.properties
            path: log4j-console.properties
```

Lampiran 12 : File deployment Flink di Kubernetes Engine (Lanjutan)

```
taskmanager-session-deployment.yaml (skenario 1)

apiVersion: apps/v1
kind: Deployment
metadata:
  name: flink-taskmanager
spec:
  replicas: 3
  selector:
    matchLabels:
      app: flink
      component: taskmanager
  template:
    metadata:
      labels:
        app: flink
        component: taskmanager
    spec:
      containers:
        - name: taskmanager
          image: apache/flink:1.17.1-scala_2.12
          args: ["taskmanager"]
          ports:
            - containerPort: 6122
              name: rpc
            - containerPort: 6125
              name: query-state
          livenessProbe:
            tcpSocket:
              port: 6122
            initialDelaySeconds: 30
            periodSeconds: 60
          volumeMounts:
            - name: flink-config-volume
              mountPath: /opt/flink/conf/
          securityContext:
            runAsUser: 9999 # refers to user _flink_ from official flink image,
change if necessary
      volumes:
        - name: flink-config-volume
          configMap:
            name: flink-config
            items:
              - key: flink-conf.yaml
                path: flink-conf.yaml
              - key: log4j-console.properties
                path: log4j-console.properties
```

Lampiran 12 : File deployment Flink di Kubernetes Engine (Lanjutan)

```
taskmanager-session-deployment.yaml (skenario 2)

apiVersion: apps/v1
kind: Deployment
metadata:
  name: flink-taskmanager
spec:
  replicas: 5
  selector:
    matchLabels:
      app: flink
      component: taskmanager
  template:
    metadata:
      labels:
        app: flink
        component: taskmanager
    spec:
      containers:
        - name: taskmanager
          image: apache/flink:1.17.1-scala_2.12
          args: ["taskmanager"]
          ports:
            - containerPort: 6122
              name: rpc
            - containerPort: 6125
              name: query-state
          livenessProbe:
            tcpSocket:
              port: 6122
            initialDelaySeconds: 30
            periodSeconds: 60
          volumeMounts:
            - name: flink-config-volume
              mountPath: /opt/flink/conf/
          securityContext:
            runAsUser: 9999 # refers to user _flink_ from official flink image,
change if necessary
      volumes:
        - name: flink-config-volume
          configMap:
            name: flink-config
            items:
              - key: flink-conf.yaml
                path: flink-conf.yaml
              - key: log4j-console.properties
                path: log4j-console.properties
```

Lampiran 13: Kode Konfigurasi docker-compose Kafka Cluster

```

version: "3.3"

services:
  zookeeper:
    restart: always
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    image: confluentinc/cp-zookeeper:latest
    ports:
      - "2181:2181/tcp"
    volumes:
      - ./zookeeper/data:/data
      - ./zookeeper/data/datalog:/datalog

  kafka:
    restart: always
    environment:
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
        PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS:
        PLAINTEXT://kafka:29092,PLAINTEXT_HOST://34.128.127.171:9092
      KAFKA_BROKER_ID: 1
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 3
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_JMX_HOSTNAME: 34.128.127.171
      KAFKA_JMX_PORT: 9999
    image: confluentinc/cp-kafka:latest
    user: root
    ports:
      - "9092:9092"
      - "9999:9999"
    volumes:
      - ./kafka/data/kafka-1:/var/lib/kafka/data
    depends_on:
      - zookeeper

  kafka-2:
    restart: always
    environment:
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
        PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka-2:29092,PLAINTEXT_HOST://34.128.127.171:9093
      KAFKA_BROKER_ID: 2
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 3
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_JMX_HOSTNAME: 34.128.127.171
      KAFKA_JMX_PORT: 9999
    image: confluentinc/cp-kafka:latest
    user: root

```

Lampiran 13: Kode Konfigurasi docker-compose Kafka Cluster (Lanjutan)

```
ports:
  - "9093:9093"
  - "9998:9999"
volumes:
  - ./kafka/data/kafka-2:/var/lib/kafka/data
depends_on:
  - zookeeper
kafka-3:
  restart: always
  environment:
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
      PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka-3:29092,PLAINTEXT_HOST://34.128.127.171:9094
      KAFKA_BROKER_ID: 3
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 3
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_JMX_HOSTNAME: 34.128.127.171
      KAFKA_JMX_PORT: 9999
  image: confluentinc/cp-kafka:latest
  user: root
  ports:
    - "9094:9094"
    - "9997:9999"
  volumes:
    - ./kafka/data/kafka-3:/var/lib/kafka/data
depends_on:
  - zookeeper

kafka-schema-registry:
  restart: always
  image: confluentinc/cp-schema-registry:5.4.0
  hostname: kafka-schema-registry
  ports:
    - "8081:8081"
  environment:
    SCHEMA_REGISTRY_KAFKASTORE_BOOTSTRAP_SERVERS:
      PLAINTEXT://kafka:29092,PLAINTEXT://kafka-2:29092,PLAINTEXT://kafka-3:29092
      SCHEMA_REGISTRY_KAFKASTORE_CONNECTION_URL: zookeeper:2181
      SCHEMA_REGISTRY_HOST_NAME: kafka-schema-registry
      SCHEMA_REGISTRY_LISTENERS: http://kafka-schema-registry:8081

kafka-ui:
  image: provectuslabs/kafka-ui
  container_name: kafka-ui
  ports:
    - "9080:8080"
  restart: always
  environment:
    KAFKA_CLUSTERS_0_NAME: local
    KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS:
      PLAINTEXT://kafka:29092,PLAINTEXT://kafka-2:29092,PLAINTEXT://kafka-3:29092
      KAFKA_CLUSTERS_0_SCHEMAREGISTRY: http://kafka-schema-registry:8081/
```

Lampiran 14: Contoh Kode Konfigurasi KTable pada Servis Trip

```

@Configuration
@EnableKafka
@EnableKafkaStreams
@Component
@Slf4j
public class TripServiceStream {
    @Autowired
    StreamsBuilderFactoryBean factoryBean;

    @Bean(name =
KafkaStreamsDefaultConfiguration.DEFAULT_STREAMS_CONFIG_BEAN_NAME)
    public KafkaStreamsConfiguration kafkaStreamsConfiguration() {
        Map<String, Object> props = new HashMap<>();
        props.put(APPLICATION_ID_CONFIG, "trip-streams");
        props.put(GROUP_ID_CONFIG, "trip-streams-group-id");
        props.put(BOOTSTRAP_SERVERS_CONFIG, bootstrapAddress);
        props.put(DEFAULT_KEY_SERDE_CLASS_CONFIG,
Sedes.String().getClass().getName());
        props.put(DEFAULT_VALUE_SERDE_CLASS_CONFIG, SpecificAvroSerde.class);
        props.put(SCHEMA_REGISTRY_URL_CONFIG, schemaRegistryUrl);
        props.put(STATE_DIR_CONFIG, "./data/trip-service/store");

        return new KafkaStreamsConfiguration(props);
    }

    @Autowired
    @Bean
    public KTable<String, AirQualityProcessedSchema>
buildPipelineAirQuality(StreamsBuilder streamsBuilder) {
        Map<String, Object> props = new HashMap<>();
        props.put(SCHEMA_REGISTRY_URL_CONFIG, schemaRegistryUrl);
        Serde<String> stringSerde = Serdes.String();
        stringSerde.configure(props, true);
        SpecificAvroSerde<AirQualityProcessedSchema> avroSerde = new
        SpecificAvroSerde<>();
        avroSerde.configure(props, false);

        return streamsBuilder
            .table(AIR_QUALITY_COMPACTED_TOPIC, Consumed.with(stringSerde,
avroSerde), Materialized.as("air-quality-state-store"));
    }

    public AirQualityProcessedSchema getAirQuality(String id) {
        log.info("GET AIR QUALITY" + id);
        KafkaStreams kafkaStreams = factoryBean.getKafkaStreams();
        assert kafkaStreams != null;
        ReadOnlyKeyValueStore<String, AirQualityProcessedSchema> amounts =
kafkaStreams
            .store(StoreQueryParameters.fromNameAndType("air-quality-state-
store", QueryableStoreTypes.keyValueStore()));
        return amounts.get(id);
    }
}

```

Lampiran 15: Tabel Perhitungan Evaluasi Kano

Riwayat Kualitas Udara (F1)		disfunctional (tidak ada)				
		Suka	Mengharap kan	Netral	Dapat menoleransi	Tidak suka
functional (ada)	Suka	0	5	9	10	14
	Mengharapkan	0	0	7	9	1
	Netral	0	0	9	3	0
	Dapat menoleransi	0	0	0	0	0
	Tidak suka	0	0	0	0	0

Pencarian Kota (F2)		disfunctional (tidak ada)				
		Suka	Mengharap kan	Netral	Dapat menoleransi	Tidak suka
functional (ada)	Suka	0	4	6	8	27
	Mengharapkan	0	0	9	4	5
	Netral	0	0	4	0	0
	Dapat menoleransi	0	0	0	0	0
	Tidak suka	0	0	0	0	0

Prediksi Kualitas Udara (F3)		disfunctional (tidak ada)				
		Suka	Mengharap kan	Netral	Dapat menoleransi	Tidak suka
functional (ada)	Suka	0	5	5	18	15
	Mengharapkan	0	0	5	10	1
	Netral	0	0	6	1	0
	Dapat menoleransi	0	0	1	0	0
	Tidak suka	0	0	0	0	0

Lampiran 15: Tabel Perhitungan Evaluasi Kano (Lanjutan)

Rekomendasi Kegiatan (F4)		disfunctional (tidak ada)				
		Suka	Mengharap kan	Netral	Dapat menoleransi	Tidak suka
functional (ada)	Suka	0	4	3	19	16
	Mengharapkan	0	1	7	3	0
	Netral	0	1	12	0	1
	Dapat menoleransi	0	0	0	0	0
	Tidak suka	0	0	0	0	0

Artikel (F5)		disfunctional (tidak ada)				
		Suka	Mengharap kan	Netral	Dapat menoleransi	Tidak suka
functional (ada)	Suka	0	3	9	8	8
	Mengharapkan	0	0	10	6	1
	Netral	0	1	16	4	0
	Dapat menoleransi	0	0	1	0	0
	Tidak suka	0	0	0	0	0

Push-Notification (F6)		disfunctional (tidak ada)				
		Suka	Mengharap kan	Netral	Dapat menoleransi	Tidak suka
functional (ada)	Suka	0	1	5	14	14
	Mengharapkan	0	1	7	5	0
	Netral	0	0	8	1	0
	Dapat menoleransi	0	2	2	2	0
	Tidak suka	1	0	2	1	1

Lampiran 15: Tabel Perhitungan Evaluasi Kano (Lanjutan)

Ringkasan Informasi Perjalanan (F7)		disfunctional (tidak ada)				
		Suka	Mengharap kan	Netral	Dapat menoleransi	Tidak suka
functional (ada)	Suka	0	4	8	9	18
	Mengharapkan	0	1	8	8	1
	Netral	0	1	9	0	0
	Dapat menoleransi	0	0	0	0	0
	Tidak suka	0	0	0	0	0

Riwayat Perjalanan (F8)		disfunctional (tidak ada)				
		Suka	Mengharap kan	Netral	Dapat menoleransi	Tidak suka
functional (ada)	Suka	0	3	2	10	18
	Mengharapkan	0	0	13	6	2
	Netral	0	1	8	3	0
	Dapat menoleransi	0	0	1	0	0
	Tidak suka	0	0	0	0	0

Jumlah Poin (F9)		disfunctional (tidak ada)				
		Suka	Mengharap kan	Netral	Dapat menoleransi	Tidak suka
functional (ada)	Suka	0	4	7	11	26
	Mengharapkan	1	0	3	6	2
	Netral	0	0	7	0	0
	Dapat menoleransi	0	0	0	0	0
	Tidak suka	0	0	0	0	0

Lampiran 15: Tabel Perhitungan Evaluasi Kano (Lanjutan)

Riwayat Penukaran Poin (F10)		disfunctional (tidak ada)				
		Suka	Mengharap kan	Netral	Dapat menoleransi	Tidak suka
functional (ada)	Suka	0	2	7	9	24
	Mengharapkan	0	1	6	3	4
	Netral	0	0	8	1	0
	Dapat menoleransi	0	0	1	0	0
	Tidak suka	1	0	0	0	0

Daftar dan Detail Kupon Tersedia (F11)		disfunctional (tidak ada)				
		Suka	Mengharap kan	Netral	Dapat menoleransi	Tidak suka
functional (ada)	Suka	0	2	8	5	35
	Mengharapkan	0	1	7	3	2
	Netral	0	0	4	0	0
	Dapat menoleransi	0	0	0	0	0
	Tidak suka	0	0	0	0	0

Daftar dan Detail Kupon Dimiliki (F12)		disfunctional (tidak ada)				
		Suka	Mengharap kan	Netral	Dapat menoleransi	Tidak suka
functional (ada)	Suka	0	2	6	5	37
	Mengharapkan	1	1	4	2	4
	Netral	0	1	4	0	0
	Dapat menoleransi	0	0	0	0	0
	Tidak suka	0	0	0	0	0