Penerapan *Choreography Message Broker* untuk Transaksi Data Berbasis *Asynchronous Restful*

Desendo Imanuel[1], Maria Nila Anggia Rini[2], Budi Susanto[3]
Informatika, Universitas Kristen Duta Wacana
Jl. Dr. Wahidin Sudirohusodo No.5-25 Yogyakarta
desendo.imanuel@ti.ukdw.ac.id
nila@ti.ukdw.ac.id
budsus@ti.ukdw.ac.id

***Abstract*—Currently, hospitals under the Christian Foundation for Public Health (Yayasan Kristen untuk Kesehatan Umum - YAKKUM) operate information systems independently. This implies that each hospital builds and manages its information system independently without systematic involvement with other hospitals. This triggers the emergence of heterogeneous data, necessitating data integration. In enterprise-scale applications, integration issues among application system components will increase with the growing needs and complexity of the application. One way to address this is by implementing middleware with a choreography approach as an intermediary between services. The system implements Axon Server as a message broker configured with an event handler in tracking processor mode. Each service has its own database supported by the implementation of infrastructure using Docker. The system is tested based on scenarios testing the reliability of data transactions between services in rollback or compensating transactions, choreography, and event consumption. Test results indicate that all testing scenarios were successfully executed even when services underwent a reboot during the testing process. Additionally, testing with unit tests, component tests, and integration tests successfully completed 46 test cases per 46 total test cases for the User Service and 27 test cases per 27 total test cases for the Auth Service. However, the test results also revealed anomalies or bugs when processing data transactions based on the saga pattern.**

Intisari—Saat ini, rumah sakit yang berada di bawah Yayasan Kristen untuk Kesehatan Umum (YAKKUM) menjalankan sistem informasi secara independen yang berarti masing-masing rumah sakit membangun dan mengelola sistem informasinya secara mandiri tanpa adanya keterlibatan secara sistematis dengan rumah sakit lainnya. Hal tersebut memicu munculnya sifat heterogen pada data sehingga diperlukannya integrasi data. Pada aplikasi skala *enterprise*, masalah integrasi antar komponen sistem aplikasi akan bertambah seiring dengan meningkatnya kebutuhan dan kompleksitas aplikasi tersebut. Salah satu cara mengatasi hal tersebut adalah mengimplementasikan *middleware* dengan pendekatan *choreography* sebagai media penengah antar *service*. Sistem mengimplementasikan Axon Server sebagai *message broker* yang dikonfigurasi dengan *event handler* dengan mode *tracking processor*. Setiap *service* memiliki *database* masing-masing yang didukung dengan penerapan infrastruktur menggunakan Docker. Sistem diuji berdasarkan skenario pengujian keandalan transaksi data antar *service* pada *roll back* atau *compensating transaction*, *choreography*, dan *event consumption*. Hasil pengujian menyatakan semua skenario pengujian berhasil diuji bahkan ketika *service* melakukan *reboot* di tengah-tengah proses pengujian. Di samping itu, pengujian dengan *unit test*, *component test*, *integration test* berhasil menyelesaikan 46 kasus tes per 46 total kasus tes pada *User Service* dan 27 kasus tes per 27 total kasus tes pada *Auth Service*. Namun, hasil pengujian juga menunjukkan adanya kejanggalan atau *bug* saat memproses transaksi data berbasis *saga pattern*.

Kata Kunci—message broker, microservice, choreography, event-driven, container, docker, axon, keycloak, rest, api

I. PENDAHULUAN

Yayasan Kristen untuk Kesehatan Umum (YAKKUM), merupakan sebuah lembaga sosial gerejawi yang bergerak di bidang kesehatan, saat ini memiliki rumah sakit yang tersebar di beberapa tempat seperti: Yogyakarta, Semarang, Surakarta (Solo), Purwokerto, Purworejo, dan Parakan [1]. Saat ini, sistem informasi yang berada di bawah yayasan YAKKUM menjalankan sistem informasi secara mandiri yang berarti masyarakat atau pasien yang sebelumnya sudah pernah melakukan pendaftaran dan memiliki jejak rekam medis pada satu rumah sakit tertentu hanya menyimpan data pasien terkait pada sistem informasi rumah sakit tersebut saja. Hal tersebut memicu munculnya sifat heterogen atau keheterogenan pada data yang digunakan pada sistem informasi terdistribusi dan diperlukan melakukan integrasi data untuk mengatasi hal tersebut [2].

Pada pengembangan aplikasi skala *enterprise*, masalah integrasi antar komponen sistem aplikasi akan muncul dan bertambah seiring dengan meningkatnya kompleksitas aplikasi yang dibangun. Menurut Mathias Weske, *Business Process Management* [2], masalah ini akan menciptakan $N \times N$ *problem* atau meningkatnya sifat ketergantungan antar service berupa *data coupling* di mana N merupakan jumlah komponen aplikasi yang akan diintegrasikan. Masalah tersebut biasanya lebih sering muncul pada aplikasi *enterprise* yang dibangun dengan pendekatan *point-to-point integration* di mana masing-masing komponen sistem aplikasi dapat secara langsung berinteraksi dan dibangun dengan basis data secara independen. Salah satu cara mengatasi hal tersebut adalah menggunakan *middleware* atau komponen perangkat lunak dalam menjembatani komunikasi antar komponen sistem aplikasi atau *services*. Dalam implementasinya, digunakan sistem arsitektur *microservices* dengan memanfaatkan pendekatan *choreography* berbasis *event-driven*. Ide utama dari implementasi *choreography* adalah untuk mengurangi sifat *tight coupling* serta

meningkatkan sifat *high cohesion* terhadap satu atau lebih *service* dengan memanfaatkan *message broker* sebagai perantara komunikasi antar *service*. Setiap *service* memiliki peran sebagai *producer* atau *consumer*, sedangkan *message broker* memiliki peran penting dalam menyampaikan suatu proses transaksi yang bersifat *asynchronous* [3]. Dengan demikian, sifat ketergantungan tidak lagi berada pada *service* melainkan berpusat pada *message broker*.

Makalah ini membahas implementasi *choreography* pada aplikasi *backend* yang dibangun berdasarkan arsitektur *microservice*. Keberhasilan penerapan Axon Server sebagai *message broker* atau *middleware* dalam menangani masalah integrasi data yang melibatkan lebih dari satu service menjadi masalah utama dalam penelitian ini. Sedangkan tujuan utama dari penelitian ini adalah menerapkan konsep *choreography* menggunakan *message broker* pada transaksi data berbasis *asynchronous* yang melibatkan lebih dari satu *service*.

II. Landasan Teori

A. Microservice

Microservice merupakan istilah yang diterapkan pada bidang ilmu komputer dalam menerapkan pengembangan sistem perangkat lunak yang berfokus pada tujuan atau *goal-oriented* dalam memenuhi tujuan *replaceability* terhadap modul-modul atau komponen aplikasi, dan ditujukan untuk dapat beradaptasi dengan ideal menjadi *big system* yang terdiri dari modul-modul kecil. *Microservice* memiliki karakteristik *modularized microservice architecture*, *cohesive microservice architecture*, dan *systematized microservice architecture* [4].

Sedangkan Chris Richardson [5] menyebutkan karakteristik yang dimiliki *microservice*, yaitu *Scale cube* yang dapat didefinisikan melalui 3 cara, yaitu:

- *X-axis scaling*: Mendistribusikan *request* kepada satu atau lebih entitas *service* yang identik.
- *Z-axis scaling*: Jika *x-axis* bekerja pada level infrastruktur maka *z-axis* bekerja pada level aplikasi atau secara *logical*.
- *Y-axis scaling*: Menekankan pada kompleksitas pengembangan aplikasi yang mengutamakan fungsionalitas dan didasarkan oleh penguraian kebutuhan fungsional hingga menjadi sekumpulan *service* yang kecil.

B. Choreography

Choreography atau koreografi merupakan arsitektur *microservice* yang memiliki sifat kemandirian yang sangat tinggi. Hal tersebut dikarenakan konsep komunikasi *asychronous* yang diterapkannya memungkinkan sistem hanya beroperasi jika terjadi suatu *event*. Konsep utama dari *choreography* ada pada metode komunikasi yang digunakan, yaitu semua metode komunikasi (*input* dan *output*) dilakukan melalui *event stream*. Entitas yang menerima *input* disebut sebagai *consumer*, sedangkan *producer* disebut sebagai entitas yang mengeluarkan *output* [3].

C. Message Broker

Message broker atau *broker* merupakan perangkat *middleware* yang berfungsi sebagai perantara untuk melakukan komunikasi dari satu aplikasi/*service* ke aplikasi/*service* lainnya. Penerapan *message broker* membutuhkan protokol komunikasi yang disebut *messaging* atau *message queuing*. Protokol tersebut merupakan protokol komunikasi satu arah yang disampaikan secara *asynchoronous* dan melibatkan *message broker* sebagai penengah [6].

D. Saga

Saga pattern merupakan suatu metode untuk melakukan *sequence of local transactions* atau transaksi lokal secara berurutan pada setiap *service* yang menjadi bagian dari rantai transaksi tersebut. *Pattern* ini menjalankan setiap transaksi dan melakukan *commit* data sebagai *single atomic transaction* [5].

E. Aggregate

Aggregate pattern merupakan sebuah objek yang terdiri dari sekumpulan entitas. Dari sekumpulan entitas tersebut, terdapat satu entitas yang disebut sebagai *aggregate root* yang berfungsi sebagai media untuk menjaga konsistensi *state* dari objek tersebut. *Pattern* ini digunakan oleh Axon untuk menjaga *state consistency* pada komponen *Command Model* yang digunakan ketika melakukan perubahan data pada *aggregate* [7].

F. Domain Driven Design (DDD)

Domain Driven Design (DDD) merupakan *pattern* yang digunakan untuk merancang pengembangan aplikasi yang berdasarkan pada *domain model*. Terminologi *domain* yang digunakan adalah sebagai sekumpulan informasi atau pengetahuan yang digunakan untuk menyelesaikan masalah-masalah pada konteks tertentu atau lingkungan di mana suatu model bisnis diterapkan. *Pattern* ini mengedepankan penerapan konsep *subdomain* dan *bounded context* untuk menganalisa dan mengidentifikasi permasalahan yang spesifik [5].

G. Command-Query Responsibility Segregation (CQRS)

Command-Query Responsibility Segregation atau CQRS merupakan *pattern* yang menekankan pemisahan komponen *command model* untuk melakukan perubahan *state* dan komponen *query model (view models* atau *projections)* untuk melihat/mengambil informasi tentang *state*. *State* tersebut merepresentasikan kondisi atau *state* terhadap data yang disimpan oleh aplikasi [7].

H. Event Sourcing

Event sourcing merupakan salah satu *pattern* yang digunakan untuk memodelkan dan menyimpan *states* sebagai *aggregate*. *Aggregate* tersebut kemudian disimpan sebagai *sequence of events*. *Pattern* tersebut berkaitan erat dengan penerapan CQRS [5]. Cara kerja *event sourcing* adalah dengan melihat suatu perubahan yang terjadi pada sistem sebagai suatu kejadian atau *event* yang terjadi di masa lampau. *Event* tersebut disimpan secara berurutan pada suatu penyimpanan yang disebut sebagai *event store* yang bersifat

JUTEI Edisi Volume. 8 No.1 April 2024
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.81.321

immutable serta dikhususkan hanya untuk menyimpan dan mencatat *events* [7].

I. Axon Framework

Axon Framework merupakan *framework* yang bersifat *open source* dan ditujukan untuk membangun aplikasi berbasis *microservice* yang menekankan pada penerapan prinsip-prinsip seperti *Domain-Driven Design* (DDD), *Command-Query Responsibility Segregation* (CQRS), dan *Event Sourcing* [7].

J. Docker

Docker merupakan platform yang memanfaatkan teknologi virtualisasi yang disebut *container* untuk menjalankan aplikasi dengan lingkungan yang terisolasi baik secara jaringan maupun media penyimpanan [8]. Pada konteks *testing* terkhusus *integration test*, Chris Richardson [5] merekomendasikan untuk menggunakan Docker sebagai sarana menjalankan *database*. Hal tersebut berkaitan erat dengan implementasi *database per service pattern* atau satu *database* per masing-masing *service*.

K. Spring Boot

Spring Boot merupakan salah satu *framework* lanjutan dari Spring Framework yang dikembangkan oleh tim Spring. *Framework* ini bersifat *open source* dan ditujukan untuk mengembangkan aplikasi berbasis bahasa pemrograman Java [9]. *Framework* ini digunakan untuk membangun aplikasi web secara RESTful pada API menggunakan format JSON.

Selain itu, Spring Data JPA (Java Persistence API) diterapkan sebagai salah satu teknologi ORM (*Object Relational Mapping*) berbasis Spring Framework yang secara spesifik digunakan untuk memodelkan data yang bersifat relasional ke dalam basis data atau *database* [10]. *Framework* ini digunakan untuk melakukan konfigurasi *database* yang terintegrasi dengan konfigurasi *view models* atau *projections* pada penerapan CQRS terkhusus pada *query model*.

L. Keycloak Server

Keycloak Server merupakan salah satu *identity providers* yang bersifat *open source*. Keycloak memiliki fitur yang sangat banyak, di antaranya *Single-Sign On* (SSO), OAuth 2.0, *Social Login*, dan OpenID Connect. Dalam implementasinya, Keycloak menggunakan token yang diperoleh setelah menyelesaikan proses autentikasi. Token ini digunakan sebagai metode pengaksesan aplikasi terproteksi berbasis REST [11].

III. PENELITIAN TERKAIT

Makalah ini berkenaan dengan beberapa penelitian terdahulu dalam mengimplementasi *message broker*—terlepas dari arsitektur sistem yang diterapkan. Perbandingan penelitian dimuat ke dalam tabel I sebagai berikut.

TABEL I
Komparasi Penelitian (a)

1.	Judul	Modelling the business structure of a digital health ecosystem [12]
	Penulis	Marcos-Pablos, García-Holgado, & García-Peñalvo, 2019
	Hasil	• *Software ecosystem* yang cenderung bermasalah hingga diberhentikan; • Kurang matangnya proposal yang diajukan terkhusus pada peran aktor-aktor yang terlibat; • *Business Process Model and Notation* (BPMN) dirasa sangat cocok untuk menggambarkan proses bisnis yang melibatkan lebih dari satu partisipan.
	Komparasi Penelitian Sekarang	Menerapkan BPMN dalam visualisasi desain alur program.
2.	Judul	Comparison of Event Choreography and Orchestration Techniques in Microservice Architecture [13]
	Penulis	Rudrabhatla, 2018

	Hasil	• Arsitektur *microservice* menimbulkan banyak masalah baru, salah satunya *distributed transaction*.
		• Jumlah *microservice*, yang juga menentukan kompleksitas dari *distributed transaction,* dapat digunakan dalam pemilihan keputusan implementasi teknik *saga pattern*.
		• Dalam perbandingannya:
		  • *choreography* akan sangat kompleks untuk diimplementasi tetapi memiliki capaian performa lebih baik;
		  • *orchestration* sangat diuntungkan dalam kasus *complex transaction* tetapi memiliki performa lebih lambat.

	Komparasi Penelitian Sekarang	Menerapkan *saga pattern* untuk beberapa API yang membutuhkan proses transaksi yang panjang/lama.
3.	Judul	University Notification Subscription System using Amazon Web Service [14]
	Penulis	Malik, et al., 2018

	Hasil	• *Software ecosystem* yang cenderung bermasalah hingga diberhentikan;
		• Dalam implementasi sistem *pub-sub* di sebuah universitas, disebutkan dapat menangani masalah: *delivery notification unguaranteed*, *decreased performance*, *redundant data*, dan *inflexible data*.
		• Jenis *message pattern* yang digunakan:
		  • *Content*: *Filtering pattern* pada konten pesan;
		  • *Topic*: Nama *message bus*;
		  • *Type*: Nama *class* pada pesan.

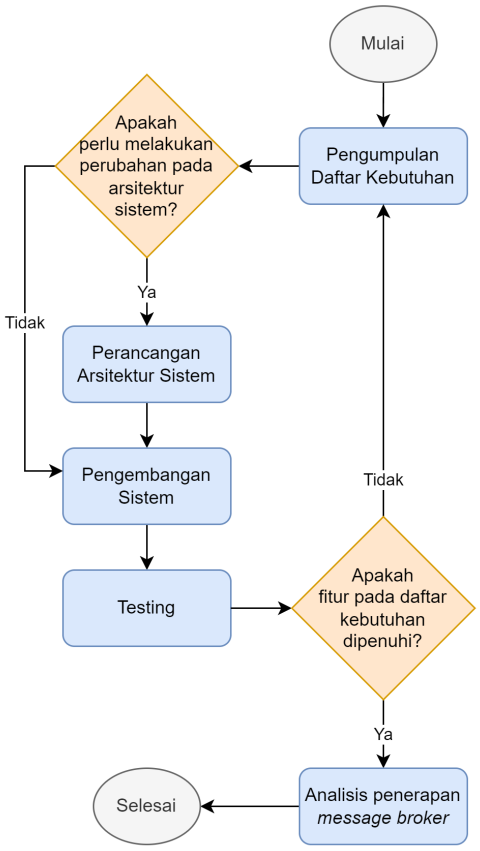
	Komparasi Penelitian Sekarang	Menerapkan klasifikasi *message pattern* berbasis CQRS (*Command and Query Responsibility Segregation*): *command*, *query*, dan *event*.

4.	Judul	Performance Analysis of RESTful API and RabbitMQ for Microservice Web Application [15]
	Penulis	Hong, Yang, & Kim, 2018
	Hasil	• RESTful API memiliki performa yang terbilang cepat tetapi memiliki besar kemungkinan terjadinya *error* menyebabkan *request* yang tidak dapat direspons hingga sistem mencapai batas kemampuan performa. • Penerapan *message broker* sebagai *middleware* terbilang lebih lambat dibanding RESTful API tetapi dikatakan tidak mendapat *error* dalam pengujiannya.
	Komparasi Penelitian Sekarang	Menerapkan *rollback/compensating transaction* untuk menangani terjadinya *error* pada alur program.

IV. METODOLOGI PENELITIAN

A. Alur Penelitian

Proses penelitian dimulai dari pengumpulan daftar kebutuhan, perancangan arsitektur sistem, pengembangan sistem, dan *testing*. Proses penelitian dilakukan secara sirkular hingga semua fitur pada daftar kebutuhan dipenuhi. Setelah itu dilakukan analisis terhadap hasil penerapan *message broker*.



Gambar. 1. Diagram Alur Penelitian

Diagram alur penelitian pada Gambar 1 di atas dijabarkan sebagai berikut.

1) Pengumpulan Daftar Kebutuhan

Pada tahap pertama Penulis akan melakukan studi literasi guna memberikan fondasi dasar mengenai penelitian yang akan dilakukan dan melakukan riset mengenai penelitian terdahulu milik orang lain yang serupa dengan penelitian ini. Penulis merangkum daftar kebutuhan dan memuatnya dalam bentuk dokumen. Dokumen-dokumen tersebut di antaranya sebagai berikut.

a. *Use Case* Diagram
b. *Use Case* Deskriptif
c. *API and Code Design*
d. Diagram Proses Bisnis (BPMN) dan *Choreography*
e. Diagram skema basis data

2) Perancangan Arsitektur Sistem

Pada tahap ini Penulis akan merancang dan mengimplementasi desain arsitektur sistem aplikasi. Penulis menuangkan desain arsitektur sistem ke dalam dokumen Arsitektur Sistem dan mulai membangun arsitektur sistem aplikasi menyesuaikan dengan dokumen daftar kebutuhan.

3) Pengembangan Sistem

Setelah tahap perancangan arsitektur sistem dilewati maka tahap pengerjaan pengembangan sistem dimulai dan pengerjaannya akan mengacu pada dokumen daftar kebutuhan.

4) Testing

Tahap *testing* ditujukan untuk menguji kebenaran fungsionalitas dari sistem aplikasi yang dibangun. Tahap ini akan terus dilakukan hingga semua daftar kebutuhan dipenuhi dan selesai dikerjakan.

5) Analisis Penerapan Message Broker

Pada tahap ini Penulis akan melakukan analisis dan evaluasi terhadap sistem aplikasi berdasarkan *testing*.

B. Use Case Diagram

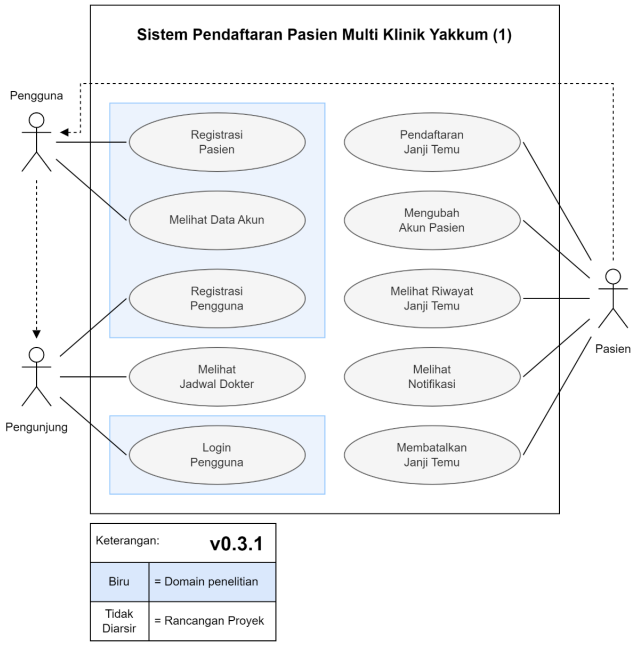
Diagram *use case* pada Gambar 2 dirancang berdasarkan dokumen *Use Case Design*. Pada penelitian ini diimplementasikan 3 aktor sebagai berikut.

a. Pengunjung/*Visitor*
   - Pengunjung dapat melakukan permintaan pengiriman kode *One Time Password* (OTP) dengan nomor telepon sebagai alamat destinasi.
   - Pengunjung dapat melakukan verifikasi nomor telepon.
   - Pengunjung dapat melakukan registrasi sebagai Pengguna.

b. Pengguna/*User*
   - Pengguna dapat melakukan registrasi sebagai Pasien pada suatu rumah sakit teregistrasi.
   - Pengguna dapat melihat detail informasi data akun miliknya pribadi.

c. Pasien/*Patient*
   - Pasien dapat melihat detail informasi data akun Pasien di rumah sakit mana ia diregistrasikan.

JUTEI Edisi Volume. 8 No.1 April 2024
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.81.321



Gambar. 2 *Use Case Diagram*

1) [UC-1-2] Registrasi Pengguna

TABEL II
[UC-1-2] Registrasi Pengguna

System	Sistem Pendaftaran Pasien Multi Klinik
Use Case	[UC-1-2] Registrasi Pengguna
Summary	Pengunjung melakukan registrasi Pengguna
Actor	Pengunjung
Pre-Condition	• [UC-1-3] Meminta Kode OTP via SMS Menggunakan Nomor Telepon • [UC-1-4] Memverifikasi Kode OTP via SMS Menggunakan Nomor Telepon
Main Sequence	1. Pengunjung menginisiasi *request* registrasi Pengguna 2. Sistem menerima *request* registrasi pengguna 3. Sistem mengambil data *User* dari *database* dan mencocokkan nomor KTP & nomor telepon dengan *request* 4. Sistem membuat pengguna baru dan menyimpannya ke *database* 5. Sistem mengembalikan *response* dengan kode status 2xx
Extensions	3.1. Jika data yang dicocokkan tidak valid atau terdapat duplikat data maka sistem akan langsung mengembalikan *response* yang berisikan informasi *error* 4xx 3.2. Jika proses pengambilan data dari *database* gagal maka sistem akan langsung mengembalikan *response* dengan kode status 5xx 4.1. Jika proses pembaruan *database* gagal maka sistem akan langsung mengembalikan *response* dengan kode status 5xx
Post-Condition	Pengunjung berhasil melakukan registrasi sebagai Pengguna. Pengunjung dapat *login* ke sistem.

2) [UC-1-3] Meminta Kode OTP via SMS Menggunakan Nomor Telepon

TABEL III
[UC-1-3] Meminta Kode OTP via SMS Menggunakan Nomor Telepon

System	Sistem Pendaftaran Pasien Multi Klinik
Use Case	[UC-1-3] Meminta Kode OTP via SMS Menggunakan Nomor Telepon
Summary	Pengunjung meminta kode OTP via SMS menggunakan nomor telepon kepada sistem
Actor	Pengunjung
Pre-Condition	
Main Sequence	1. Pengunjung menginisiasi *request* meminta kode OTP via SMS menggunakan nomor telepon 2. Sistem menerima *request* meminta kode OTP via SMS menggunakan nomor telepon 3. Sistem mengambil data riwayat OTP dari *database* berdasarkan nomor telepon 4. Sistem men-*generate* kode OTP berupa numerik dan mengirimkannya melalui *messaging service* sesuai nomor telepon pada *request* 5. Sistem mengembalikan *response* dengan kode status 2xx
Extensions	2.1. Selain nomor telepon, atribut pada *request* dapat berisikan alamat email 3.1. Jika nomor telepon tidak valid/cocok maka sistem akan mencatat nomor telepon ke *database* riwayat OTP 3.2. Jika proses pengambilan data dari *database* gagal maka sistem akan langsung mengembalikan *response* dengan kode status 5xx

	4.1. Jika *request* pengiriman kode OTP melalui *messaging service* gagal maka sistem akan langsung mengembalikan *response* dengan kode status 5xx
Post-Condition	Sistem mengirim kode OTP kepada nomor telepon yang tertera pada *request* via SMS

3) [UC-1-4] Memverifikasi Kode OTP via SMS Menggunakan Nomor Telepon

TABEL IV
[UC-1-4] Memverifikasi Kode OTP via SMS Menggunakan Nomor Telepon

System	Sistem Pendaftaran Pasien Multi Klinik
Use Case	[UC-1-4] Memverifikasi Kode OTP Menggunakan Nomor Telepon
Summary	Pengunjung mengirim kembali kode OTP yang diterima via SMS kepada sistem untuk diverifikasi dan divalidasi
Actor	Pengunjung
Pre-Condition	• [UC-1-3] Meminta Kode OTP via SMS Menggunakan Nomor Telepon
Main Sequence	1. Pengunjung menginisiasi *request* memverifikasi kode OTP menggunakan nomor telepon 2. Sistem menerima *request* mengirim kembali kode OTP yang didapat via SMS kepada sistem untuk diverifikasi dan divalidasi 3. Sistem mengambil data riwayat OTP dari *database* berdasarkan atribut yang ada pada *request* 4. Sistem mencatat atribut pada *request* dan memperbarui *database* 5. Sistem mengembalikan *response* dengan kode status 2xx
Extensions	2.1. Selain nomor telepon, atribut pada *request* dapat berisikan alamat email 3.1. Jika nomor telepon dan kode OTP tidak valid/cocok maka sistem akan langsung mengembalikan *response* dengan kode status 4xx 3.2. Jika proses pengambilan data dari *database* gagal maka sistem akan langsung mengembalikan *response* dengan kode status 5xx 4.1. Jika proses pembaruan *database* gagal maka sistem akan langsung mengembalikan *response* dengan kode status 5xx
Post-Condition	Kode OTP terkirim kepada nomor telepon yang tertera pada *request* via SMS

4) [UC-1-5] Memverifikasi Login Menggunakan Nomor Telepon

TABEL V
[UC-1-5] Memverifikasi Login Menggunakan Nomor Telepon

System	Sistem Pendaftaran Pasien Multi Klinik
Use Case	[UC-1-5] Memverifikasi Login Menggunakan Nomor Telepon
Summary	Pengunjung melakukan *login* ke sistem sebagai Pengguna menggunakan nomor telepon dan kode OTP yang diterima via SMS
Actor	Pengunjung
Pre-Condition	• [UC-1-3] Meminta Kode OTP via SMS Menggunakan Nomor Telepon • [UC-1-4] Memverifikasi Kode OTP via SMS Menggunakan Nomor Telepon
Main Sequence	1. Pengunjung menginisiasi *request login* ke sistem menggunakan nomor telepon 2. Sistem menerima *request login* ke sistem menggunakan nomor telepon 3. Sistem mengambil data riwayat OTP dari *database* berdasarkan nomor telepon dan kode OTP pada *request* 4. Sistem mencatat atribut pada *request* dan memperbarui *database* 5. Sistem mengembalikan *response* berisi *token* dengan kode status 2xx
Extensions	3.1 Jika nomor telepon dan kode OTP tidak valid/cocok maka sistem akan langsung mengembalikan *response* dengan kode status 4xx 3.2 Jika proses pengambilan data dari *database* gagal maka sistem akan langsung mengembalikan *response* dengan kode status 5xx 4.1. Jika proses pembaruan *database* gagal maka sistem akan langsung mengembalikan *response* dengan kode status 5xx

JUTEI Edisi Volume. 8 No.1 April 2024
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.81.321

Post-Condition	Pengunjung berhasil *login* ke sistem menggunakan nomor telepon dan teridentifikasi sebagai Pengguna

5) [UC-2-1] Registrasi Pasien Baru

TABEL VI
[UC-2-1] Registrasi Pasien Baru

System	Sistem Pendaftaran Pasien Multi Klinik
Use Case	[UC-2-1] Registrasi Pasien Baru
Summary	Pengguna melakukan registrasi Pasien baru pada suatu Rumah Sakit
Actor	Pengguna
Pre-Condition	• [UC-1-2] Registrasi Pengguna • [UC-1-5] Memverifikasi Login Menggunakan Nomor Telepon
Main Sequence	1. Pengguna menginisiasi *request* registrasi Pasien baru 2. Sistem menerima *request* registrasi Pasien baru 3. Sistem mengambil data Pengguna dari *database* berdasarkan atribut yang ada pada *request* 4. Sistem mencatat atribut pada *request*, membuat data Pasien, memperbarui *database*, dan menyimpan dokumen pada *request* 5. Sistem mengembalikan *response* berisikan data *medicalRecordId* dengan kode status 2xx
Extensions	2.1. Jika *token* tidak valid/cocok maka sistem akan langsung mengembalikan *response* dengan kode status 4xx 3.1. Jika data Pengguna tidak valid/ditemukan maka sistem akan langsung mengembalikan *response* dengan kode status 4xx 3.2. Jika proses pengambilan data dari *database* gagal maka sistem akan langsung mengembalikan *response* dengan kode status 5xx 4.1. Jika proses pembaruan *database* gagal maka sistem akan langsung mengembalikan *response* dengan kode status 5xx
Post-Condition	Pengguna berhasil melakukan registrasi sebagai Pasien

6) [UC-2-3] Melihat Data Akun

TABEL VII
[UC-2-3] Melihat Data Akun

System	Sistem Pendaftaran Pasien Multi Klinik
Use Case	[UC-2-3] Melihat Data Akun
Summary	Pengguna melihat data profil/akun
Actor	Pengguna
Pre-Condition	• [UC-1-2] Registrasi Pengguna • [UC-1-5] Memverifikasi Login Menggunakan Nomor Telepon
Main Sequence	1. Pasien menginisiasi *request* melihat data profil/akun 2. Sistem menerima *request* melihat data profil/akun 3. Sistem mengambil data Pasien berdasarkan atribut yang ada pada *request* 4. Sistem mengembalikan *response* berisi informasi mengenai data Pengguna dengan kode status 2xx
Extensions	2.1. Jika *token* tidak valid/cocok maka sistem akan langsung mengembalikan *response* dengan kode status 4xx 3.1. Jika data Pengguna tidak valid/cocok maka sistem akan langsung mengembalikan *response* dengan kode status 4xx 3.2. Jika proses pengambilan data dari *database* gagal maka sistem akan langsung mengembalikan *response* dengan kode status 5xx
Post-Condition	Pengguna melihat data profil/akun miliknya pribadi

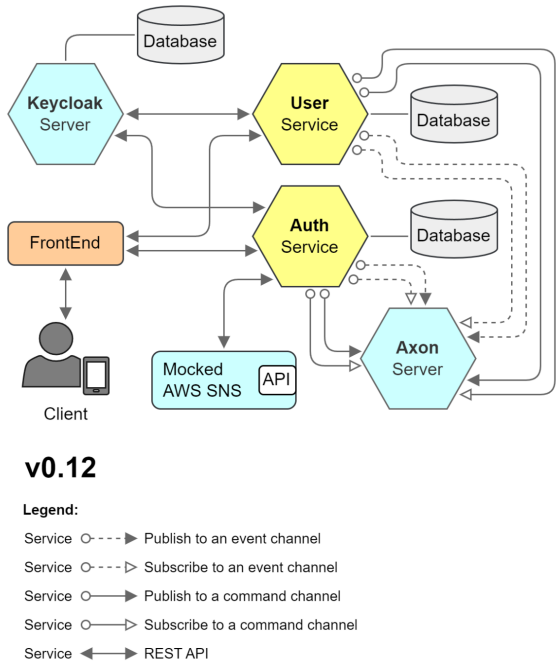
C. Arsitektur Sistem

Metode komunikasi antar *service* akan dilakukan secara *asynchronous* dengan menerapkan *messaging pattern* atau *event-driven*. Dalam menangani *event*, setiap *service* yang berwarna kuning akan berperan sebagai *publisher* atau *consumer*, atau *publisher* sekaligus *consumer*. *Service* tersebut tidak berinteraksi secara langsung dan tidak saling tahu bahwa *service* tersebut berinteraksi dengan *service* lainnya. Setiap *service* akan memiliki basis data (*database*) dan berfokus pada pekerjaannya masing-masing dengan lingkup domain yang spesifik dan khusus. Penerapan Axon Server sebagai *message broker* merupakan kunci utama dalam arsitektur sistem, sebagai *middleware,* yang bertugas meneruskan komunikasi data dari *publisher* ke *subscriber*.

Penamaan *service* pada arsitektur sistem ini diambil berdasarkan konteks yang didapat pada desain basis data, yaitu segmentasi kepemilikan yang sesuai dengan jenis data yang dikelompokkan berdasarkan domain data tersebut.

Berdasarkan metode-metode tersebut, diambil kesimpulan seperti berikut dan divisualisasikan seperti pada Gambar 3.

- *Auth Service* berfokus pada autentikasi dan otorisasi, yaitu *role* dan *permission* yang dibutuhkan untuk mengakses layanan tertentu serta mengirimkan kode *one time password* (OTP) untuk memverifikasi pengguna yang ingin membuat akun sebagai pasien.
- *User Service* berfokus pada manajemen akun pasien, yaitu data personal, dokumen pribadi, dan urusan registrasi, yaitu pendaftaran akun baru, dan pendaftaran janji temu dengan dokter spesialis khusus.



Gambar. 3. Arsitektur Sistem

D. *Perancangan dan Pengujian Sistem*

Pengujian sistem dibuat dan didesain berdasarkan dokumen kebutuhan fungsional dan non-fungsional. Buku Microservices Patterns oleh Chris Richardson [5] menjadi salah satu acuan untuk melakukan pengujian. Pengujian pada sistem dikategorisasikan sebagai berikut.

- *Unit test* untuk menguji unit terkecil seperti *class* atau entitas/objek.
- *Component test* untuk menguji perilaku sebuah *service* beserta dependensi-nya (unit/modul apa saja yang digunakannya).
- *Integration test* untuk menguji integrasi antar *service/microservice*, *message broker*, basis data, dan layanan pihak ketiga.

Untuk menguji penerapan transaksi data, dibuat skenario pengujian sebagai berikut.

- Skenario *roll back* atau *compensating transaction*. Skenario ini menguji penerapan *roll back* pada proses *choreography* transaksi data. Pengujian ini akan dengan sengaja menggagalkan salah satu transaksi dari sebuah proses bisnis. Pada saat pengujian, Penulis akan mengamati terjadinya perubahan *state* sebelum dan sesudah *roll back* dieksekusi oleh sistem.
- Skenario *choreography*. Pengujian ini ditujukan untuk mengamati penerapan konsep *choreography* sebagai metode komunikasi data pada komponen *event handler* dan pengiriman kode OTP yang dilakukan secara *asynchronous* antara *Auth Service* dan *User Service* khususnya pada proses registrasi akun Pengguna.
- Skenario mengonsumsi *event*: Penerapan transaksi menggunakan *command* kepada *aggregate* akan mengakibatkan perubahan *state* yang diidentifikasikan dengan sebuah *event* yang merepresentasikan kejadian tersebut. Skenario ini ditujukan untuk mengamati proses konsumsi *event* sebagai penerapan konsep *choreography* dalam melakukan integrasi data dari *User Service* ke *Auth Service*.

Pada proses pengujian sistem berdasarkan skenario, setiap *service* akan diuji dengan konfigurasi berikut.

- Menjalankan dan menguji aplikasi secara langsung melalui Visual Studio Code. Pengujian ini dilakukan untuk mengamati proses transaksi data yang terjadi pada saat aplikasi memproses *request* melalui REST API.
- Untuk mengetahui keberhasilan penerapan REST, dilakukan pengujian REST API menggunakan aplikasi Postman.

V. IMPLEMENTASI DAN ANALISIS SISTEM

A. *Infrastruktur*

Infrastruktur dibangun menggunakan teknologi *container* dari Docker berupa *docker-compose* [8]. Terdapat empat *container* yang menjadi infrastruktur dasar yaitu *Auth Service*, *User Service*, Keycloak, dan Axon Server seperti pada Gambar 3. *Container* phpMyAdmin hanya untuk melakukan pengecekan secara visual data yang tersimpan pada *database*.

Pada penelitian ini, sebuah Axon Server [7] dianggap sebagai *message broker*. *Broker* tersebut diberi konfigurasi *event handler* dengan mode *tracking processor*. Setiap modul aplikasi atau *service* akan menerapkan satu *physical database* dan menggunakan Axon Framework [7] untuk dapat berkomunikasi dengan Axon Server. Untuk manajemen akun pengguna dan keamanan, diimplementasikan Keycloak Server [11]. Setiap *service* akan terhubung dalam satu jaringan yang sama melalui konfigurasi jaringan yang dinamakan *multiklinik-network*.

B. *Penulisan Program*

1) *Aggregate*

Aggregate sendiri merupakan sekumpulan objek yang teridentifikasi sebagai satu unit. Untuk dapat memodifikasi *aggregate* diperlukan sebuah *command* yang salah satu atributnya dianotasikan dengan @TargetAggregateIdentifier. Sebuah *aggregate* dibangun dari serangkaian peristiwa yang sudah terjadi. Gambar 4 merupakan implementasi dari *User Aggregate*. File ini menyimpan *aggregate* lain sebagai bagian darinya seperti informasi identitas, kontak, alamat, dan informasi nomor rekam medis yang terasosiasikan pada Pengguna.

JUTEI Edisi Volume. 8 No.1 April 2024
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.81.321

```java
@Aggregate
public class UserAggregate {

    @AggregateIdentifier
    private Long idUser;

    @AggregateMember
    private AddressAggregate address;
    @AggregateMember
    private ContactAggregate contact;
    @AggregateMember
    private IdentityAggregate identity;
    @AggregateMember
    private List<PatientAggregate> patientIdentities;

    public UserAggregate() {}

    @CommandHandler
    public UserAggregate(
        RegisterUserCommand registerUserCommand
    ) {
        UserRegistrationEvent userRegistrationEvent = new UserRegistrationEvent();
        BeanUtils.copyProperties(registerUserCommand, userRegistrationEvent);

     userRegistrationEvent.setIdUser(registerUserCommand.getIdUser());


     AggregateLifecycle.apply(userRegistrationEvent);
    }

    @CommandHandler
    public void handle(
        CreateUserCommand createUserCommand
    ) {
        UserCreatedEvent userCreatedEvent = new UserCreatedEvent();
        BeanUtils.copyProperties(createUserCommand, userCreatedEvent);

        AggregateLifecycle.apply(userCreatedEvent);
    }

    @EventSourcingHandler
    public void on(UserRegistrationEvent userRegistrationEvent) {
        this.idUser = userRegistrationEvent.getIdUser();
    }

}
```

Gambar. 4. *User Aggregate*

2) Command

Command digunakan untuk melakukan modifikasi pada sumber daya pada sistem. Setiap *command* mengharuskan memiliki sebuah atribut yang dianotasikan @TargetAggregateIdentifier dengan tujuan untuk membuat sebuah *aggregate* baru atau membangun ulang (*retrieve*) *aggregate* dari kumpulan *events* yang tersimpan pada *event store*. Konfigurasi dan pengiriman *command* dapat dilakukan pada *file* Java mana pun, namun Penulis menerapkannya pada *file-file* yang diakhiri dengan penamaan berikut:

- .*CommandController\.java
- .*CommandServiceImpl\.java
- .*EventsHandler\.java

Gambar 5 merupakan salah satu penerapan *command* yang berfungsi untuk melakukan perubahan *state* yang mendeskripsikan entitas Pengguna.

```java
@Data
@NoArgsConstructor
public class CreateUserCommand {

    @TargetAggregateIdentifier
    private Long idUser;

    private Long idIdentity;
    private Long idContact;
    private Long idAddress;
    private Long idOtp;

    private String phoneNumber;

}
```

Gambar. 5. CreateUserCommand.java

Command CreateUserCommand diterapkan seperti pada Gambar 6 untuk melakukan registrasi Pengguna. *Command* tersebut akan dikirim kepada Axon Server sebagai *message broker* untuk kemudian diarahkan kepada *command handler* seperti gambar 4.1 pada *constructor UserAggregate* (baris 19 hingga 25).

```java
    @Override
    public StandardResponseModel getPatientProfiles(Long idUser) {
        StandardResponseModel response = new StandardResponseModel();

        UserDetailsQuery userDetailsQuery = new UserDetailsQuery(idUser);
        PatientProfilesQuery patientProfilesQuery = new PatientProfilesQuery(idUser);
        try {
            UserEntity userEntity = queryGateway
                .query(userDetailsQuery, ResponseTypes.instanceOf(UserEntity.class))
                .join();

            JSONArray patients = queryGateway
                .query(patientProfilesQuery, ResponseTypes.instanceOf(JSONArray.class))
                .join();


   response.setStatusCode(HttpStatus.OK.value());
            response.putData("identity", CustomParser.parseUserIdentity(userEntity.getIdentity()).toMap());
            response.putData("contact", CustomParser.parseUserContact(userEntity.getContact()).toMap());
            response.putData("address", CustomParser.parseUserAddress(userEntity.getAddress()).toMap());
            response.putData("patients", patients.toList());

            return response;
        } catch (HttpStatusCodeException e) {
            throw e;
        } catch (Exception e) {
            throw e;
        }
    }
```

Gambar. 6. Fungsi registerUser() Pada *File* UsersCommandServiceImpl.java

3) Event

Penerapan *event* ditujukan sebagai kejadian pada sistem yang sudah terjadi di masa lampau. Hal ini dapat berupa perubahan pada sistem, seperti UserCreatedEvent pada Gambar 7 yang menandakan terjadinya penambahan Pengguna pada sistem; terdapat suatu kejadian yang wajib untuk dicatat pada *event store*, seperti *UserRegistrationEvent* yang memberitahukan sistem bahwa ada suatu *request* yang mengharuskan sistem untuk mencatatnya/menginjeksi terlebih dahulu pada *saga*. Hal tersebut didukung dengan proses pendaftaran Pengguna yang diidentifikasikan sebagai suatu proses transaksi yang panjang seperti: melakukan validasi; membuat *aggregate*, membuat *identity*, *contact*, *address*, dan *user* sebagai satu transaksi.

```java
@Data
@NoArgsConstructor
public class UserCreatedEvent {

    private Long idUser;
    private Long idIdentity;
    private Long idContact;
    private Long idAddress;
    private Long idOtp;

}
```

Gambar. 7. UserCreatedEvent.java

4) Query

Implementasi *query* ditujukan untuk melakukan pengambilan data pada *projection database*. Untuk setiap *query*, terdapat tepat satu fungsi yang dianotasikan @QueryHandler untuk menangani *query* tersebut seperti UserDetailsQuery pada Gambar 8. Konfigurasi dan pengiriman *query* dapat dilakukan pada *file* Java mana pun, namun Penulis menerapkannya pada *file-file* yang diakhiri dengan penamaan berikut.

- .*QueryController\.java
- .*QueryServiceImpl\.java
- .*QueryHandler\.java

Penggunaan *query* diintegrasikan dengan komponen *query handler* seperti pada Gambar 9. Fungsi *getPatientProfiles(Long idUser)* tersebut digunakan untuk melakukan pencarian pada *view models* atau *projections* menggunakan *database*. Integrasi tersebut dapat dilihat pada Gambar 10 yang memiliki fungsi untuk melakukan *query* pada *database* untuk mencari entitas dengan kriteria *idUser*.

```java
@Data
@NoArgsConstructor
public class UserDetailsQuery {

    private Long idUser;
    private String medicalRecordId;

    public UserDetailsQuery(String medicalRecordId) {
        this.medicalRecordId = medicalRecordId;
    }

    public UserDetailsQuery(Long idUser) {
        this.idUser = idUser;
    }

}
```

Gambar. 8. UserDetailsQuery.java

```java
    @Override
    public StandardResponseModel getPatientProfiles(Long idUser) {
        StandardResponseModel response = new StandardResponseModel();

        UserDetailsQuery userDetailsQuery = new UserDetailsQuery(idUser);
        PatientProfilesQuery patientProfilesQuery = new PatientProfilesQuery(idUser);
        try {
            UserEntity userEntity = queryGateway
                .query(userDetailsQuery, ResponseTypes.instanceOf(UserEntity.class))
                .join();

            JSONArray patients = queryGateway
                .query(patientProfilesQuery, ResponseTypes.instanceOf(JSONArray.class))
                .join();


         response.setStatusCode(HttpStatus.OK.value());
            response.putData("identity", CustomParser.parseUserIdentity(userEntity.getIdentity()).toMap());
            response.putData("contact", CustomParser.parseUserContact(userEntity.getContact()).toMap());
            response.putData("address", CustomParser.parseUserAddress(userEntity.getAddress()).toMap());
            response.putData("patients", patients.toList());

            return response;
        } catch (HttpStatusCodeException e) {
            throw e;
        } catch (Exception e) {
            throw e;
        }
    }
```

Gambar. 9. Fungsi getPatientProfiles() Pada *File* UsersQueryServiceImpl.java

```java
    @QueryHandler
    public UserDto fetchUser(
        UserQuery userQuery
    ) {
        UserEntity userEntity = userRepository.findById(userQuery.getIdUser().longValue());

        if (userEntity == null) {
            throw new UserIsNotRegisteredException();
        }

        UserDto userDto = CustomMapper.mapToDto(userEntity);

        return userDto;
    }
```

Gambar. 10. Fungsi fetchUser() Pada *File* UsersQueryHandler.java

5) Komunikasi Data

Penerapan komunikasi data berbasis *choreography* pada sistem mengandalkan *aggregate*, *command*, *event*, dan *query*. Konfigurasi *message pattern* berada di "axon.eventhandling.processors" pada *file* application.properties seperti Gambar 11 dan Gambar 12. Pilihan mode *tracking* membantu *event handler* untuk dapat memantau proses konsumsi *events*. Mode *tracking* menggunakan *tracking tokens* yang disimpan pada *database* sehingga lebih dapat diandalkan ketika terjadi kegagalan

JUTEI Edisi Volume. 8 No.1 April 2024
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.81.321

mendadak pada *service*. Konfigurasi ini dapat dilihat sebagai konfigurasi/ketertarikan *consumer* terhadap *events* apa saja yang ia butuh dengan melakukan *subscribe*. Berikut rincian konfigurasi konfigurasi *tracking processor* tersebut.

- *User Service* tertarik pada 7 jenis *event*:
  - *user-group*: *events* mengenai entitas Pengguna (*User*)
    - UserRegistrationRejectedEvent.java
    - UserCreatedEvent.java
  - *user-identity-group*: *events* mengenai entitas Identitas Pengguna (*Identity*)
    - IdentityCreatedEvent.java
    - UserRegistrationRejectedEvent.java
  - *user-contact-group*: *events* mengenai entitas Kontak Pengguna (*Contact*)
    - ContactCreatedEvent.java
    - UserRegistrationRejectedEvent.java
  - *user-address-group*: *events* mengenai entitas Alamat Pengguna (*Address*)
    - AddressCreatedEvent.java
    - UserRegistrationRejectedEvent.java
  - *patient-group*: *events* mengenai entitas Pasien (*Patient*)
    - PatientRegisteredEvent.java
  - *otp-group*: *events* mengenai entitas Kode OTP (*OTP*)
    - UserOtpVerifiedEvent.java
    - UserOtpVerifyRejectedEvent.java
  - *hospital-group*: *events* mengenai entitas Rumah Sakit (*Hospital*)
    - HospitalProjectionPersistedEvent.java
- *Auth Service* hanya membutuhkan 3 jenis *event*:
  - *otp-group*: *events* mengenai entitas Kode OTP (*OTP*)
    - UserOtpVerifyEvent.java
    - OtpCreatedEvent.java
    - OtpRequestedEvent.java
    - OtpVerifiedEvent.java
    - UserOtpVerifiedEvent.java
    - UserProjectionPersistedEvent.java
    - OtpCreatedEvent.java
    - OtpRequestedEvent.java
  - *user-group*: *events* mengenai entitas Pengguna (*User*)
  - *patient-group*: *events* mengenai entitas Pasien (*Patient*)
- PatientRegisteredEvent.java

Konfigurasi Tracking Processor pada konfigurasi kredensial *User Service*.

```
...
axon.eventhandling.processors.user-group.mode=tracking
axon.eventhandling.processors.user-identity-group.mode=tracking
axon.eventhandling.processors.user-contact-group.mode=tracking
axon.eventhandling.processors.user-address-group.mode=tracking
axon.eventhandling.processors.patient-group.mode=tracking
axon.eventhandling.processors.otp-group.mode=tracking
axon.eventhandling.processors.hospital-group.mode=tracking
...
```

Gambar. 11. Konfigurasi *Tracking Processor* Pada Konfigurasi Kredensial *User Service*

```
...
axon.eventhandling.processors.otp-group.mode=tracking
axon.eventhandling.processors.user-group.mode=tracking
axon.eventhandling.processors.patient-group.mode=tracking
...
```

Gambar. 12. Konfigurasi *Tracking Processor* Pada Konfigurasi Kredensial *Auth Service*

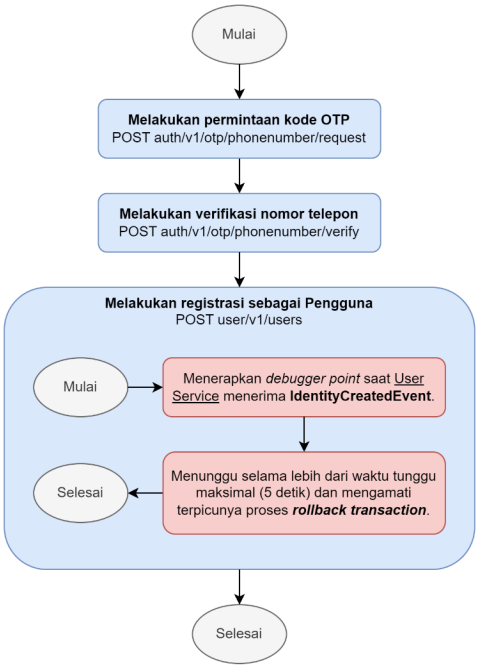
6) Library

Penulis menerapkan implementasi *core library* pada sistem untuk menyimpan modul-modul *events* dan *commands*. Axon menerapkan identifikasi unik untuk modul tersebut dan menekankan penerapan *domain driven design*. Sehingga konfigurasi dependensi tersebut harus diletakkan pada *library* khusus sebagai penengah dan lokasi penggunaan *shared modules*. Dengan demikian, konfigurasi dapat dijabarkan sebagai berikut.

- *Auth Service* bergantung kepada *Core*
- *User Service* bergantung kepada *Core*

C. Pengujian dan Analisis: Tes Skenario: Rollback/Compensating Transaction

Pengujian *rollback transaction* pada Gambar 13 ditujukan untuk menguji alur kompensasi transaksi untuk registrasi Pengunjung sebagai Pengguna.



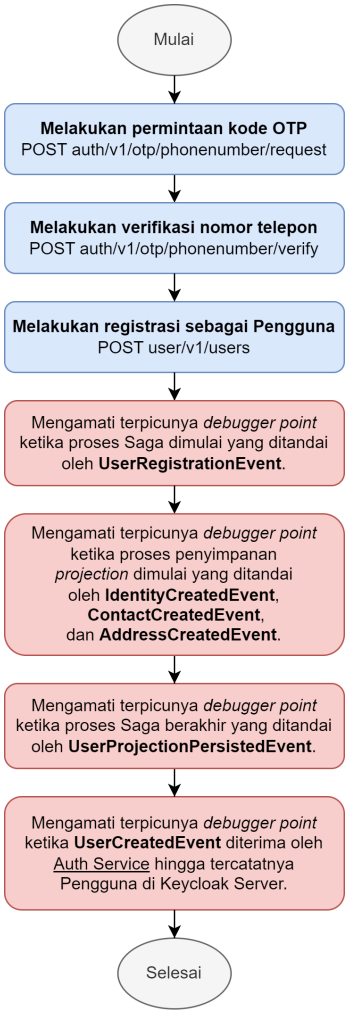
Gambar. 13. Alur Proses Tes Skenario: *Rollback/Compensating Transaction*

Berdasarkan tes skenario *roll back* atau *compensating transaction* pada Gambar 13, dapat disimpulkan bahwa pengujian terbilang efektif dalam menangani terjadinya *error* pada proses pendaftaran Pengguna (*User*) bahkan ketika melakukan *service reboot* secara paksa pada saat *service* sedang mengonsumsi *event*. Sistem dapat dengan sendirinya membatalkan proses transaksi. Selain itu, sistem dapat dengan sendirinya melakukan *roll back* pada bagian-bagian transaksi yang sebelumnya berhasil dieksekusi dan diproyeksikan pada *database*.

Pada beberapa peristiwa pengujian, diidentifikasikan beberapa hasil pengujian mengalami kegagalan dalam melakukan *roll back* atau *transaction.* Dari peristiwa tersebut, dapat terjadi *leftover entity* atau entitas yang masih tersimpan pada *projection database* yang seharusnya dihapus. Temuan tersebut terjadi secara tidak tentu sehingga Penulis tidak dapat menguji secara objektif mengenai kejanggalan tersebut. Hipotesis yang dapat diambil oleh Penulis adalah adanya kesalahan konfigurasi pada "eventhandling.processors" dan regulasi konsumsi *event*.

D. Pengujian dan Analisis: Tes Skenario: Choreography

Pengujian *choreography* pada Gambar 14 ditujukan untuk melakukan inspeksi dan analisis pada penerapan *choreography* sebagai bagian dari proses komunikasi data antara *User Service* dan *Auth Service*.



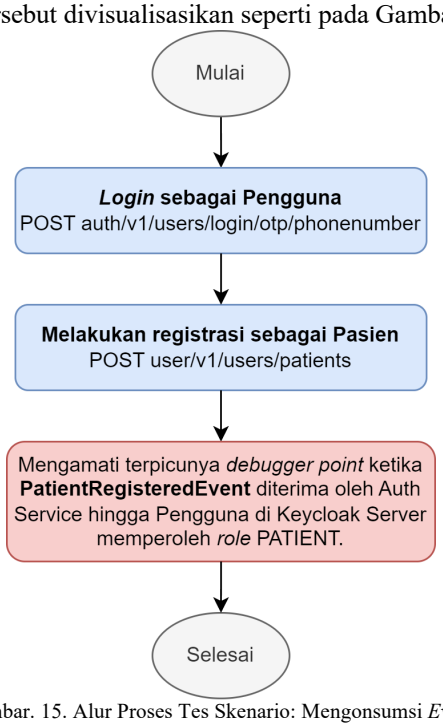
Gambar. 14. Alur Proses Tes Skenario: *Choreography*

Proses pada pendaftaran akun Pengguna menggunakan konsep *orchestrator* dan *choreography*. Implementasi *orchestrator* terletak pada *file UserRegistrationSaga.java* yang menjadi pusat pengambilan keputusan. Sedangkan penerapan *choreography* pada pengujian ini hanya diimplementasikan pada komponen *event handler* untuk melakukan proyeksi *aggregate* ke dalam *database.* Berdasarkan Gambar 26, pengujian ini dapat disebut berhasil untuk memenuhi tes skenario dengan penerapan konsep *choreography* yang diimplementasikan pada proses verifikasi kode OTP oleh *User Service* yang secara *asynchronous* dikimkan kepada Axon Server lalu diterima oleh *Auth Service* untuk kemudian melakukan perubahan pada data OTP. Di samping itu, penerapan API pada fungsionalitas registrasi akun Pengguna dianggap berhasil yang dibuktikan dengan dimulainya *UserRegistrationEvent* dan diakhiri dengan *UserProjectionPersistedEvent*. Meski demikian, tes skenario *choreography* masih memiliki *bug* pada implementasi *saga*. Penulis menemukan bahwa *bug* tersebut hanya terjadi ketika terpicunya proses pembuatan *aggregate* pada salah satu proses di *saga*.

Dalam implementasi dan pengujian pada alur registrasi Pengguna (*User*) yang mengimplementasikan *saga*, Penulis menemukan kejadian di mana *exception* berupa *NullPointerException* yang kerap kali muncul secara tidak tentu. Berikut adalah kepingan *error* kode program tersebut. Beberapa *error* pada *log* mengindikasikan kalau peristiwa ini berhubungan dengan proses pembuatan *aggregate*.

E. Pengujian dan Analisis: Tes Skenario: Mengonsumsi Event

Pengujian ini melibatkan konsumsi *event* secara berantai baik dari *User Service* ke *Auth Service* maupun sebaliknya. Proses tersebut divisualisasikan seperti pada Gambar 15.



Gambar. 15. Alur Proses Tes Skenario: Mengonsumsi *Event*

Konsumsi *event* dilakukan pada komponen yang dinamakan *event handler*. Pengujian ini. Tes skenario "mengonsumsi *event*" dapat disimpulkan berhasil untuk

JUTEI Edisi Volume. 8 No.1 April 2024
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.81.321

diterapkan. Hal tersebut dibuktikan dengan berhasilnya pengujian transaksi untuk pada *Auth Service* dalam mengonsumsi *event PatientRegisteredEvent* yang merupakan reaksi terhadap *command RegisterPatientCommand* oleh *User Service*. Proses transaksi tersebut mengindikasikan bahwa seorang Pengguna baru saja melakukan registrasi sebagai Pasien. Hal tersebut dianggap sebagai metode integrasi data untuk melakukan sinkronisasi data antara *User Service* dan *Auth Service*. Proses konsumsi *event* tersebut diimplementasikan pada komponen *event handler* untuk memproses penambahan *role PATIENT* ke Pengguna yang teregistrasi pada Keycloak Server.

F. Pengujian: Unit Test, Component Test, dan Integration Test

Pengujian dilakukan menggunakan Spring Boot *testing framework* dan H2 *in-memory database*. Konfigurasi pengujian tiap *file* mengacu pada standarisasi Struktur *Folder* seperti pada Tabel VIII berikut.

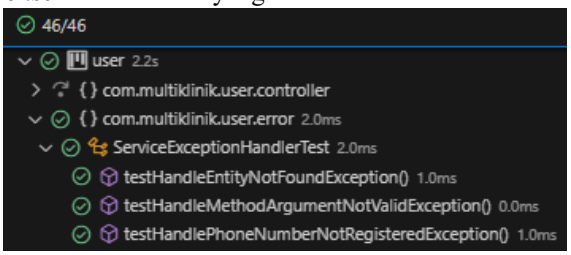
TABEL VIII
Strandar Struktur Folder

└──**auth** // nama *service*
├──**config** // konfigurasi *service*
├──**controller** // REST *endpoint*
├──**error** // konfigurasi khusus untuk menangani *error*
├──**helper** // kumpulan *utilities*
├──**model** // koleksi *data model*
│ ├──**aggregate** // entitas yang digunakan dalam menerapkan *aggregate pattern*
│ ├──**command** // model yang digunakan untuk melakukan perubahan pada *aggregate*
│ ├──**data** // model yang digunakan untuk *mapping* ke *database*
│ ├──**dto** // *data transfer* yang bersifat sementara (digunakan sebagai perantara *data model* dan *service*)
│ ├──**event** // *event* yang digunakan dalam menerapkan *event-driven*
│ ├──**query** // digunakan untuk melakukan *query* ke *projection database*
│ └──**request** // digunakan untuk menerima HTTP *body request*
├──**repository** // konfigurasi *object-relational mapper* (ORM)
├──**security** // penerapan konfigurasi keamanan
└──**service** // implementasi proses bisnis; implementasi berdasarkan dokumen BPMN Microservice Choreography
├──**command** // modifikasi terhadap data
├──**event** // respons *service* terhadap *event*
├──**interceptor** // inspeksi oleh *service* terhadap *command* sebelum di-*dispatched* ke Axon
└──**query** // *query* terhadap *projection database*

1) User Service

User Service berhasil menyelesaikan pengujian *unit, component, integration* sebanyak 46 tes dari 46 kasus tes. Pengujian *aggregate* berhasil menyelesaikan 12 tes per 12 kasus tes dari total 4 *aggregates*. Pengujian *saga* berhasil menyelesaikan 8 tes per 8 kasus tes.

Gambar 16 merupakan pengujian terhadap implementasi *exception handler* yang bertugas untuk membangun REST *response* dalam bentuk yang terformat.

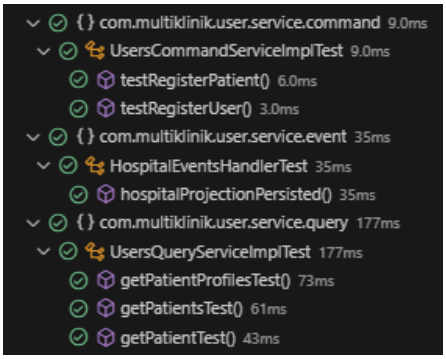


Gambar. 16. Pengujian *User Service* - ServiceExceptionHandlerTest

Gambar 17 merupakan pengujian berupa *unit test* untuk menguji apakah atribut, fungsi, dan *constructor* pada *aggregate* seperti *address*, contact, identity, patient, dan user dapat berfungsi dengan baik.

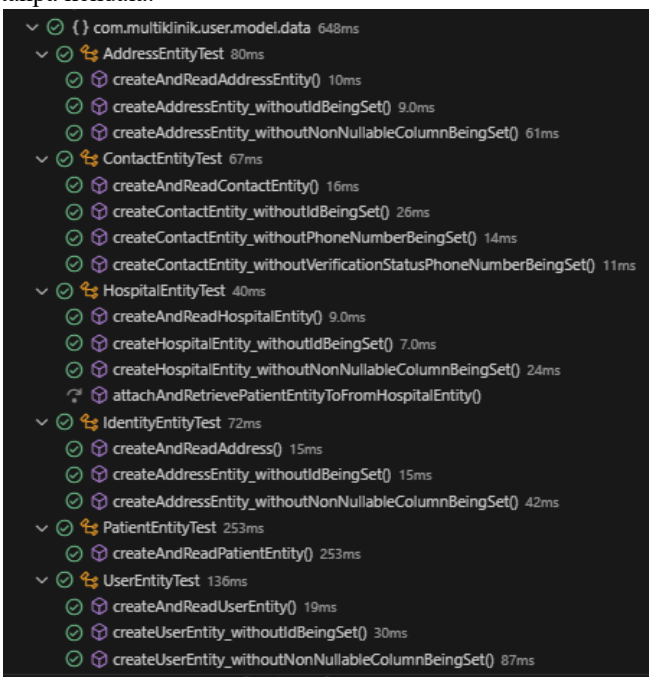
Gambar. 17. Pengujian *User Service - Aggregate*

Gambar 18 merupakan pengujian secara *integration test* untuk menguji proses bisnis yang diimplementasikan pada *file* ServiceImpl.java seperti fungsi *registerUser()* pada *file* UsersCommandServiceImpl.java pada gambar 6. Nama fungsi mendeskripsikan pengujian yang dilakukan. Pengujian ini memenuhi kebutuhan fungsional, yaitu Pengguna berhasil melakukan registrasi sebagai Pasien, Pengguna dapat melihat informasi data akun miliknya, Pasien dapat melihat informasi data akun Pasien miliknya.



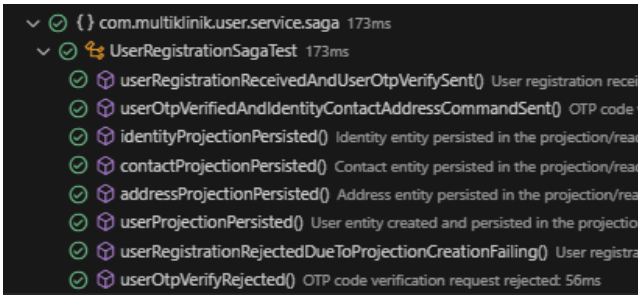
Gambar. 18. Pengujian *User Service* - *Command*, *Event*, dan *Query*

Gambar 19 merupakan pengujian *integration test* pada *persistence layer* dalam mengimplementasi *database*. Setiap entitas dikonfigurasi menggunakan H2 *in-memory database*. Pengujian ini dilakukan untuk memastikan bahwa integrasi antara aplikasi dan *database* dapat berjalan dengan baik tanpa kendala.



Gambar. 19. Pengujian *User Service* - *Data Model*

Gambar 20 merupakan keberhasilan pengujian *unit test* terhadap implementasi *saga pattern* untuk melakukan registrasi Pengguna. Pengujian dilakukan untuk memastikan bahwa setiap komponen pada *saga* dapat berfungsi dengan baik. Pengujian ini dilakukan dengan menggunakan *library* yang disediakan Axon. Pada pengujian tersebut, memenuhi kebutuhan fungsional berupa aturan penerapan nomor telepon yang terverifikasi sebagai syarat pendaftaran akun Pengguna baru.

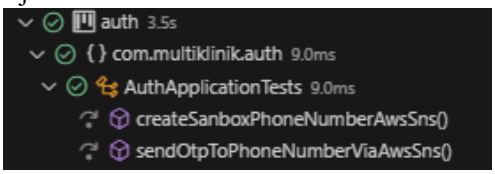


Gambar. 20. Pengujian *User Service* - *Saga*

2) Auth Service

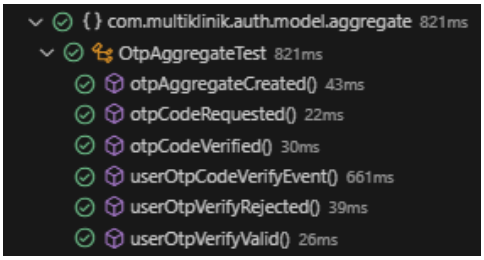
Auth Service berhasil menyelesaikan pengujian *unit, component, integration* sebanyak 27 tes per 27 kasus tes. Pengujian *aggregate* berhasil menyelesaikan 6 tes per 6 kasus tes dari total 1 *aggregate*. Pengujian Keycloak API berhasil menyelesaikan 5 tes per 5 kasus tes.

Gambar 21 merupakan pengujian *integration test* secara *mocking* pada implementasi autentikasi AWS SNS untuk mengirim kode OTP menggunakan nomor telepon sebagai alamat tujuan.



Gambar. 21. Pengujian *Auth Service* - AWS SNS

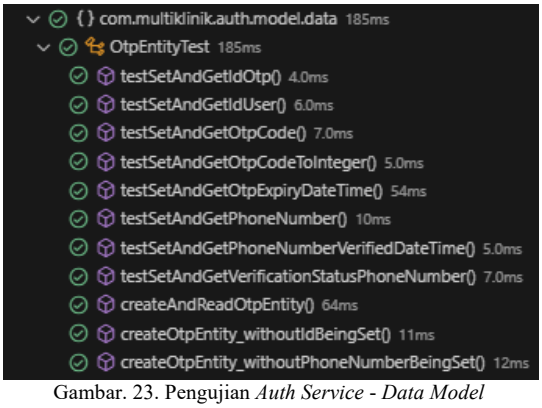
Gambar 22 merupakan pengujian terhadap *aggregate* berupa *unit test*. Pengujian ini memiliki konsep yang sama seperti pengujian *aggregate* pada *User Service*, yaitu memastikan bahwa fungsionalitas *aggregate* dapat berfungsi dengan baik.



Gambar. 22. Pengujian *Auth Service* - *Aggregate*

Gambar 23 merupakan pengujian secara *integration test* untuk menguji integrasi layanan dalam mengimplementasikan *database*. Pengujian ini dilakukan dengan melakukan operasi CRUD (*Create*, *Read*, *Update*, dan *Delete*) terhadap model entitas yang diuji. Berdasarkan pengujian ini, kebutuhan fungsional untuk memverifikasi nomor telepon menggunakan kode OTP dipenuhi.

JUTEI Edisi Volume. 8 No.1 April 2024
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.81.321



Gambar. 23. Pengujian *Auth Service - Data Model*

Gambar 24 menguji implementasi Keycloak Server pada aplikasi. Pengujian ini dilakukan secara fungsional melalui pengujian dalam memanggil REST API yang disediakan oleh Keycloak Server.

Gambar. 24. Pengujian *Auth Service - Event Handler* dan Keycloak *Service*

VI. KESIMPULAN DAN SARAN

Kesimpulan dari penelitian ini dijabarkan sebagai berikut.

1. Berdasarkan hasil pengujian, penerapan Axon Server sebagai *message broker* pada penelitian ini terbilang berhasil dengan pengecualian sebagai berikut.
    a. Pada pengujian *compensating transaction*, diambil kesimpulan bahwa integritas data pada saat melakukan transaksi data dapat dijaga yang dibuktikan dengan berhasilnya dalam menerapkan *roll back* atau *compensating transaction*. Meski demikian, program masih meninggalkan *intermittent error* pada alur program terkhusus pada beberapa skenario yang Penulis tidak dapat identifikasi lebih lanjut.
    b. Pengujian *choreography* mengindikasikan bahwa penerapan transaksi data berhasil diterapkan pada lingkungan *microservice* dengan pendekatan *choreography* pada lebih dari satu *service* (*User Service* dan *Auth Service*). Hal tersebut dibuktikan dengan keberhasilan penerapan komponen *event handler* dalam memproyeksikan data ke *projection database* dan penerapan pengiriman kode OTP yang melibatkan dua *service*, yaitu *User Service* dan *Auth Service*. Namun, program masih meninggalkan *intermittent error* yang diasumsikan pada implementasi *saga* atau *aggregate*.
    c. Pengujian terhadap *event handler* pada *User Service* dan *Auth Service* berhasil mengonsumsi *event* dengan lancar tanpa kendala. Hal ini mengindikasikan bahwa berhasilnya penerapan integrasi data dari *User Service* ke *Auth Service*. Hal tersebut juga terimplementasi pada proses pendaftaran akun Pengguna yang mana data Pengguna pada *User Service* akan digunakan oleh *Auth Service* untuk meregistrasikan Pengguna pada Keycloak Server.
2. Daftar kebutuhan fungsional terpenuhi berdasarkan pengujian *unit*, *component*, dan *integration*.
3. Daftar kebutuhan non-fungsional terpenuhi berdasarkan implementasi perancangan sistem, yaitu Axon Server sebagai *message broker*; arsitektur *microservice* yang mengadopsi konsep *choreography*; basis data relasional menggunakan MySQL; Keycloak Server dalam mengimplementasikan layanan autentikasi pada mekanisme *login*.

Saran yang dapat dipertimbangkan untuk penelitian selanjutnya adalah sebagai berikut.

1. Masih terdapat *error* atau *bug* pada proses pengujian, terkhusus pada bagian implementasi *saga pattern* untuk melakukan pendaftaran akun Pengguna. Proses pendaftaran akun Pengguna merepresentasikan penerapan *choreography* pada proses transaksi data antara *User Service* dan *Auth Service*. Dengan demikian, pembenahan atau perbaikan terhadap transaksi data tersebut merupakan hal yang penting.

UCAPAN TERIMA KASIH

Penulis mengucap syukur dan berterima kasih atas kesempatan dan dukungan yang diberikan oleh orang tua, dosen pembimbing satu, dosen pembimbing dua, dan teman-teman sehingga penelitian ini dapat terselesaikan dengan baik.

DAFTAR PUSTAKA

[1] Yayasan Kristen untuk Kesehatan Umum, "Unit Layanan," 16 Juni 2022. [Online]. Available: https://yakkum.or.id/unit-layanan/. [Diakses 16 Juni 2022].

[2] M. Weske, Business Process Management, Heidelberg: Springer Berlin, 2019.

[3] A. Bellemare, Building Event-Driven Microservices, O'Reilly Media, Inc., 2020.

[4] I. Nadareishvili, R. Mitra, M. McLarty dan M. Amundsen, Microservice Architecture: Aligning

Principles, Practices, and Culture, O'Reilly Media, Inc., 2016.

[5] C. Richardson, Microservices Patterns With Examples in Java, Manning Publications Co., 2019.

[6] L. Johansson dan D. Dossot, RabbitMQ Essentials: Build distributed and scalable applications with message queuing using RabbitMQ, Packt Publishing, 2020.

[7] AxonIQ, "Axon Reference Guide," 01 Desember 2023. [Online]. Available: https://docs.axoniq.io/reference-guide/.

[8] Docker, Inc., "Docker Overview," 01 Desember 2023. [Online]. Available: https://docs.docker.com/get-started/overview/.

[9] Spring Projects, "Spring Boot," 01 Desember 2023. [Online]. Available: https://spring.io/projects/spring-boot.

[10] Spring Projects, "Spring Data JPA," 01 Desember 2023. [Online]. Available: https://spring.io/projects/spring-data-jpa.

[11] Keycloak, "Server Administration Guide Version 22.0.5," 1 Desember 2023. [Online]. Available: https://www.keycloak.org/docs/22.0.5/server_admin/.

[12] S. Marcos-Pablos, A. García-Holgado dan F. J. García-Peñalvo, "Modelling the business structure of a digital health ecosystem," *Proceedings of the Seventh International Conference on Technological Ecosystems for Enhancing Multiculturality,* pp. 838-846, Oktober 2019.

[13] C. K. Rudrabhatla, "Comparison of Event Choreography and Orchestration Techniques in Microservice Architecture," *International Journal of Advanced Computer Science and Applications,* vol. 9, no. 8, pp. 18-22, 2018.

[14] B. H. Malik, Z. M. Dar, S. M. Kayani, M. Dar, M. H. Shafiq, I. Kabir, F. Masood, H. Zakriya dan A. Ali, "University Notification Subscription System using Amazon Web Service," *International Journal of Advanced Computer Science and Applications,* vol. 9, no. 5, pp. 349-354, 2018.

[15] X. J. Hong, H. S. Yang dan Y. H. Kim, "Performance Analysis of RESTful API and RabbitMQ for Microservice Web Application," *2018 International Conference on Information and Communication Technology Convergence (ICTC),* pp. 257-259, 2018.

JUTEI Edisi Volume. 8 No.1 April 2024
ISSN 2579-3675, e-ISSN 2579-5538
DOI 10.21460/jutei.81.321