TUGAS AKHIR - EF234801

RANCANG BANGUN KERANGKA KERJA MODUL TENANT MANAGEMENT UNTUK APLIKASI SOFTWARE AS A SERVICE

MARCELLINO MAHESA JANITRA
NRP 5025201105

Dosen Pembimbing I
Rizky Januar Akbar, S.Kom., M.Eng.
NIP 198701032014041001

Dosen Pembimbing II
Sarwosri, S.Kom. M.T
NIP 197608092001122001

Program Studi S1 Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya
2024

TUGAS AKHIR – EF234801

RANCANG BANGUN KERANGKA KERJA MODUL TENANT MANAGEMENT UNTUK APLIKASI SOFTWARE AS A SERVICE

MARCELLINO MAHESA JANITRA
NRP 5025201105

Dosen Pembimbing I
Rizky Januar Akbar, S.Kom., M.Eng.
NIP 198701032014041001

Dosen Pembimbing I
Sarwosri, S.Kom. M.T
NIP 197608092001122001

Program Studi S1 Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya
2024

<halaman ini sengaja dikosongkan>

FINAL PROJECT – EF234801

TENANT MANAGEMENT MODULE FRAMEWORK DESIGN FOR SOFTWARE AS A SERVICE

MARCELLINO MAHESA JANITRA
NRP 5025201105

Advisor
Rizky Januar Akbar, S.Kom., M.Eng.
NIP 198701032014041001

Co-Advisor
Sarwosri, S.Kom. M.T
NIP 197608092001122001

Study Program Bachelor of Informatics
Department of Informatics
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya
2024

<halaman ini sengaja dikosongkan>

LEMBAR PENGESAHAN

RANCANG BANGUN KERANGKA KERJA MODUL TENANT MANAGEMENT UNTUK APLIKASI SOFTWARE AS A SERVICE

TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
Memperoleh gelar Sarjana Komputer pada
Program Studi S-1 Teknik Informatika
Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh :
MARCELLINO MAHESA JANITRA
NRP. 5025201105

Disetujui oleh Tim Penguji Tugas Akhir:

1.	Rizky Januar Akbar, S.Kom., M.Eng.	Pembimbing
2.	Sarwosri, S.Kom. M.T	Ko-pembimbing
3.	Ratih Nur Esti Anggraini, S.Kom, M.Sc., Ph.D.	Penguji
4.	Siska Arifiani, S.Kom., M.Kom.	Penguji

SURABAYA
Juli, 2024

<halaman ini sengaja dikosongkan>

APPROVAL SHEET

TENANT MANAGEMENT MODULE FRAMEWORK DESIGN FOR SOFTWARE AS A SERVICE

FINAL PROJECT
Submitted to fulfill one of the requirements
for obtaining a degree Computer Sceince at
Undergraduate Study Program of Informatics
Department of Informatics
Faculty of Electrical and Information Technology
Institut Teknologi Sepuluh Nopember

By:
MARCELLINO MAHESA JANITRA
NRP. 5025201105

Approved by Final Project Examiner Team:

1.	Rizky Januar Akbar, S.Kom., M.Eng.	Advisor
2.	Sarwosri, S.Kom. M.T	Co-Advisor
3.	Ratih Nur Esti Anggraini, S.Kom, M.Sc., Ph.D.	Examiner
4.	Siska Arifiani, S.Kom., M.Kom.	Examiner

SURABAYA
July, 2024

<halaman ini sengaja dikosongkan>

PERNYATAAN ORISINALITAS

Yang bertanda tangan di bawah ini:

Nama mahasiswa / NRP	: Marcellino Mahesa Janitra / 5025201105
Departemen	: Teknik Informatika
Dosen Pembimbing 1 / NIP	: Rizky Januar Akbar, S.Kom., M.Eng. / 198701032014041001
Dosen Pembimbing 2 / NIP	: Sarwosri, S.Kom. M.T / 197608092001122001

Dengan ini menyatakan bahwa Tugas Akhir dengan judul "Rancang Bangun Kerangka Kerja Modul Tenant Management untuk Aplikasi Software as a Service" adalah hasil karya sendiri, bersifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.
Bilamana di kemudian hari dietemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima saksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, 26 Juli 2024

Mengetahui Dosen Pembimbing 1	Mengetahui Dosen Pembimbing 2	Mengetahui Mahasiswa
Rizky Januar Akbar, S.Kom., M.Eng.	Sarwosri, S.Kom. M.T	Marcellino Mahesa Janitra
198701032014041001	197608092001122001	5025201105

<halaman ini sengaja dikosongkan>

STATEMENT OF ORIGINALITY

The undersigned below:

Nama of student / NRP	: Marcellino Mahesa Janitra / 5025201105
Departement	: Teknik Informatika
Advisor / NIP	: Rizky Januar Akbar, S.Kom., M.Eng. / 198701032014041001
Co-Advisor / NIP	: Sarwosri, S.Kom. M.T / 197608092001122001

Hereby declare that Final Project with the title of "Tenant Management Module Framework Design for Sofware as a Service" is the result of my own work, is original, and is written by following the rules of scientific writing.
If in the future there is a discrepancy with this statement, then I am willing to accept sanctions
in accordance with the provisions that apply at Institut Teknologi Sepuluh Nopember.

Surabaya, 26 July 2024

Acknowledge Dosen Pembimbing 1	Acknowledge Dosen Pembimbing 2	Acknowledge Mahasiswa
Rizky Januar Akbar, S.Kom., M.Eng.	Sarwosri, S.Kom. M.T	Marcellino Mahesa Janitra
198701032014041001	197608092001122001	5025201105

<halaman ini sengaja dikosongkan>

RANCANG BANGUN KERANGKA KERJA MODUL TENANT MANAGEMENT UNTUK APLIKASI *SOFTWARE AS A SERVICE*

Nama Mahasiswa / NRP : Marcellino Mahesa Janitra / 5025201105

Departemen : Teknik Informatika FTEIC- ITS

Dosen Pembimbing : Rizky Januar Akbar, S.Kom., M.Eng.

Dosen Pembimbing 2 : Sarwosri, S.Kom. M.T

ABSTRAK

Model Software as a Service (SaaS) semakin marak digunakan, karena kemudahan dan fleksibilitas yang ditawarkan. Bisnis yang menggunakan model SaaS dapat menghemat biaya operasional dan menghemat waktu dalam mendirikan sistem bisnisnya. Salah satu bentuk populer dari model ini adalah SaaS multi-tenant. Satu instance dari aplikasi pelanggan (Tenant) melayani banyak tenant sekaligus. SaaS multi-tenant memiliki keuntungan diantaranya efisiensi biaya, kemudahan skala, dan pembaruan perangkat lunak yang terpusat. Hal ini membuat model SaaS multi-tenant menjadi pilihan menarik bagi banyak organisasi.

Modul Tenant Management dalam SaaS multi-tenant bertugas untuk mengelola berbagai aspek operasional dari setiap tenant, yaitu: migrasi sumber daya, aktivasi tenant, deaktivasi tenant dan berkomunikasi dengan modul lain dalam sistem.

Tugas akhir ini berhasil merancang dan mengembangkan modul Tenant Management menggunakan arsitektur CQRS (Command Query Responsibility Segregation) pada pola DDD (Domain Driven Design). Penggunaan CQRS pada DDD memudahkan pengembang untuk menggunakan kerangka kerja ini sebagai basis pembuatan modul Tenant Management, karena abstraksi dari domain bisnis sudah terdefinisi dengan jelas. Komunikasi dilakukan menggunakan EDA (*Event Driven Architecture*) yang menurunkan tingkat ketergantungan langsung antar modul. Hasil dari tugas akhir ini dibangun menggunakan bahasa Go sebagai bahasa pemrograman *backend,* GCP sebagai wadah *deployment* dan berkomunikasi menggunakan *event* dan *terraform* sebagai alat melakukan *deployment* sumber daya tenant.

Kata kunci: Multi-tenant, SaaS, Golang, GCP , Tenant Management

TENANT MANAGEMENT MODULE FRAMEWORK DESIGN FOR SOFTWARE AS A SERVICE

Student Name / NRP : **Marcellino Mahesa Janitra / 5025201105**

Departement : **Teknik Informatika FTEIC- ITS**

Advisor : **Rizky Januar Akbar, S.Kom., M.Eng.**

Advisor 2 : **Sarwosri, S.Kom. M.T**

ABSTRACT

The Software as a Service (SaaS) model is increasingly popular due to the convenience and flexibility it offers. Businesses using the SaaS model can save on operational costs and time when setting up their business systems. A popular form of this model is multi-tenant SaaS, where a single instance of an application serves multiple customers (Tenants) simultaneously. Multi-tenant SaaS has advantages such as cost efficiency, easy scalability, and centralized software updates, making it an attractive choice for many organizations.

The Tenant Management module in multi-tenant SaaS is responsible for managing various operational aspects of each tenant, such as resource migration, tenant activation, tenant deactivation, and communication with other system modules.

This final project successfully designed and developed a tenant management module using CQRS (Command Query Responsibility Segregation) architecture in a DDD (Domain Driven Design) pattern. The use of CQRS in DDD facilitates developers in using this framework as the basis for creating the Tenant Management module because the business domain abstraction is clearly defined. Communication is conducted using EDA (Event Driven Architecture), which reduces direct dependencies between modules. The outcome of this project was built using Go as the backend programming language, GCP for deployment, and event-driven communication, with Terraform used for deploying tenant resources.

Keywords: Multi-tenant, SaaS, Golang, GCP, Tenant Management

KATA PENGANTAR

Puji syukur ke hadirat Tuhan Yang Maha Esa, karena berkat kekuatan dan bimbingan-Nya penulis dapat menyelesaikan tugas akhir dengan judul "Rancang Bangun Kerangka Kerja Modul Tenant Management untuk Aplikasi Software as a Service". Penyusunan tugas akhir ini dilakukan dalam rangka memenuhi salah satu syarat untuk memperoleh gelar Sarjana Komputer pada Program Studi S-1 Teknik Informatika, Departemen Teknik Informatika, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember. Tugas akhir ini tidak dapat diselesaikan tanpa bantuan dan dukungan dari banyak pihak yang terlibat secara langsung maupun tidak langsung, sehingga penulis ingin mengucapkan terima kasih kepada:

1. Bapak Dosen Pembimbing Rizky Januar Akbar, S.Kom., M.Eng. dan Ibu Sarwosri, S.Kom. M.T. atas bimbingan, arahan, dan pengawasannya dalam proses penulisan tugas akhir ini.
2. Keluarga penulis yang selalu memberikan doa, dukungan, dan semangat dalam setiap langkah perjalanan penulis.
3. Alexander, Rafiqi, Ryo dan Putu Dhika selaku tim pendukung pengerjaan Modul eksternal.
4. Teman-teman yang selalu memberikan semangat, diskusi, dan menemani penulis hingga saat ini.
5. Pihak lain yang tidak dapat disebutkan satu-persatu oleh penulis, yang telah membantu penulis selama perkuliahan.

Penulis menyadari bahwa tugas akhir ini masih memiliki banyak kekurangan. Sehingga kritik, saran, dan masukan yang membangun sangat penulis harapkan guna perbaikan di masa yang akan datang. Akhir kata, semoga tugas akhir ini dapat memberikan manfaat dan sumbangsih bagi ilmu pengetahuan dan teknologi di bidang Teknik Informatika. Penulis berharap tugas akhir ini dapat menjadi dasar untuk penelitian dan pengembangan di masa depan.

Surabaya, 26 Juli 2024

Marcellino Mahesa Janitra
5025201105

<halaman ini sengaja dikosongkan>

DAFTAR ISI

<halaman ini sengaja dikosongkan>

DAFTAR GAMBAR/GRAFIK/DIAGRAM

DAFTAR TABEL

DAFTAR KODE SUMBER

Daftar Singkatan

DDD	Domain Driver Design
EDA	Event Driven Architecture
GCP	Google Cloud Platform
SaaS	Software as a Service
CQRS	Command Query Responsibility Segregation
IAM	Identity Access Management

<halaman ini sengaja dikosongkan>

BAB I PENDAHULUAN

1.1 Latar Belakang

Software as a Service atau SaaS merupakan model bisnis berbasis *cloud* yang populer di era IT saat ini (Aung et al., 2009). Penekanan biaya IT dengan sistem *pay as you go* oleh SaaS, serta kemudahannya dalam menyusun Sistem IT perusahaan dan dapat diakses dari mana saja melalui internet mendorong perusahaan atau individu untuk mengadopsi SaaS dalam kebutuhan bisnis mereka (Aung et al., 2009). Pada dasarnya, SaaS adalah bisnis yang menyediakan solusi untuk kebutuhan penyewa layanan, yang mudah untuk diakses dan disetup dari mana saja, untuk memaksimalkan keuntungan (Golding, 2024).

Multi-tenanitas adalah sebuah model di mana bisnis menerapkan hanya satu aplikasi yang digunakan untuk melayani banyak pelanggan, hal ini tentu saja lebih ekonomis dari pada harus mendedikasikan satu aplikasi untuk setiap pelanggan (Cai et al., 2013). Dengan saling berbagi sumber daya, bisnis dapat menghemat biaya karena sangat mungkin untuk memanajemen sumber daya serta melakukan beban kerja secara efisien melalui *requests* atau *threads* (Krebs et al., 2012), penekanan biaya ini paling efisien pada level *instance* aplikasi (Momm & Krebs, 2011). Agar aplikasi dapat memuaskan ekspetasi pelanggan, sistem bisnis harus menerapkan isolasi fungsional dan non-fungsional pada tenant, yang menimbulkan tantangan besar karena sumber daya *instance* aplikasi harus dapat dibagikan satu tenant dengan yang lain (Krebs et al., 2012). Dalam konteks ini, tenant adalah entitas atau organisasi yang menyewa layanan yang disediakan SaaS. Secara umum sebuah tenant meliputi kumpulan pelanggan yang memiliki kepentingan dalam melakukan kegiatan bisnis pada entitas atau organisasi tersebut (Bezemer & Zaidman, n.d.).

Dalam aplikasi SaaS multi-tenant dibutuhkan sistem yang dapat melayani manajemen tenant. Implementasi antarmuka layanan SaaS dipecah menjadi dua, yaitu layanan manajemen dasar dan manajemen operasional. Level layanan manajemen dasar pada umumnya dilakukan menggunakan metode CRUD (*create, read update, delete*). Manajemen operasional tenant merupakan level layanan lebih luas, yang mengatur *lifecycle* tenant. Salah satu kompenen penting lainnya adalah pembuatan *identifier* sebagai nilai global yang unik untuk sebuah tenant yang tidak memiliki keterkaitan terhadap attribut lain dari tenant. Sistem manajemen tenant juga harus dapat menyimpan *identifier* yang telah terbuat sebelumnya untuk dapat digunakan dalam mengidentifikasi tenant. Operasi manajemen tenant yang telah dibahas sebelumnya cenderung menambah kompleksitas dalam layanan manajemen tenant, namun sangat dibutuhkan untuk modul Tenant Management dalam aplikasi SaaS multi-tenant, maka dari itu developer harus memikirkan strategi yang cocok agar tidak terjadi *bottleneck* dalam sistem (Golding, 2024).

Tugas akhir ini akan menghasilkan kerangka kerja sistem manajemen tenant yang merupakan satu bagian dari aplikasi SaaS yang lebih besar. Hal ini mengharuskan modul Tenant Management untuk dapat berkolaborasi dengan modul relevan lainnya dengan mempertahankan independensi data tenant yang ada, ini berarti sumber kebenaran data tenant adalah dari modul Tenant Management dan tidak memiliki keterkaitan langsung dengan modul lain. Untuk itu diperlukan perencanaan desain sistem dengan pengembang modul lain agar dapat membangun aplikasi SaaS secara keseluruhan.

1.2 Rumusan Permasalahan

Tugas akhir ini membahas hal-hal sebagai berikut:

1. Bagaimana cara merancang kerangka kerja sistem manajemen tenant untuk aplikasi SaaS?
2. Bagaimana cara membangun kerangka kerja sistem manajemen tenant untuk aplikasi SaaS?
3. Bagaimana cara mengintegrasikan kerangka kerja modul Tenant Management dengan aplikasi SaaS secara utuh?
4. Bagaimana melakukan pengujian pada kerangka kerja modul Tenant Management untuk aplikasi SaaS?

1.3 Batasan Masalah

Tugas akhir ini memiliki Batasan sebagai berikut:

1. Hasil dari Pembangunan aplikasi SaaS berupa kerangka yang dapat dipakai dalam mengembangkan aplikasi SaaS secara utuh.
2. Pengerjaan dan pembangunan aplikasi sistem manajemen tenant ini menggunakan bahasa pemrograman Go sebagai *backend*, Next.js sebagai *frontend*, dan menggunakan *platform* GCP.

1.4 Tujuan

Tujuan dari tugas akhir adalah:

1. Merancang kerangka kerja sistem manajemen tenant untuk aplikasi SaaS.
2. Membangun kerangka kerja sistem manajemen tenant untuk aplikasi SaaS.
3. Mengintegrasikan kerangka kerja modul Tenant Management dengan aplikasi SaaS secara utuh.
4. Melakukan pengujian pada kerangka kerja modul Tenant Management untuk aplikasi SaaS.

1.5 Manfaat

Hasil tugas akhir ini diharapkan secara teoritis dapat menambah pengetahuan tentang bagaimana cara merancang dan membangun aplikasi sistem manajemen tenant untuk aplikasi SaaS, dan secara praktis hasil dari tugas akhir ini diharapkan dapat digunakan dalam mengembangkan aplikasi SaaS secara keseluruhan.

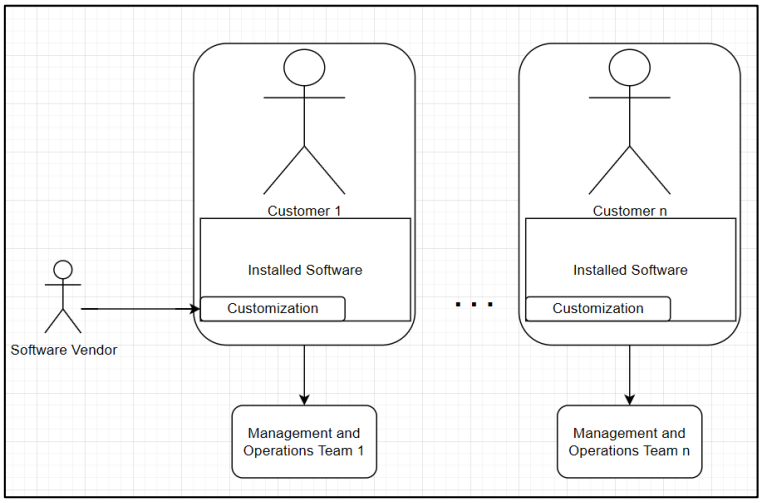
BAB II TINJAUAN PUSTAKA

2.1 Penelitian Terkait

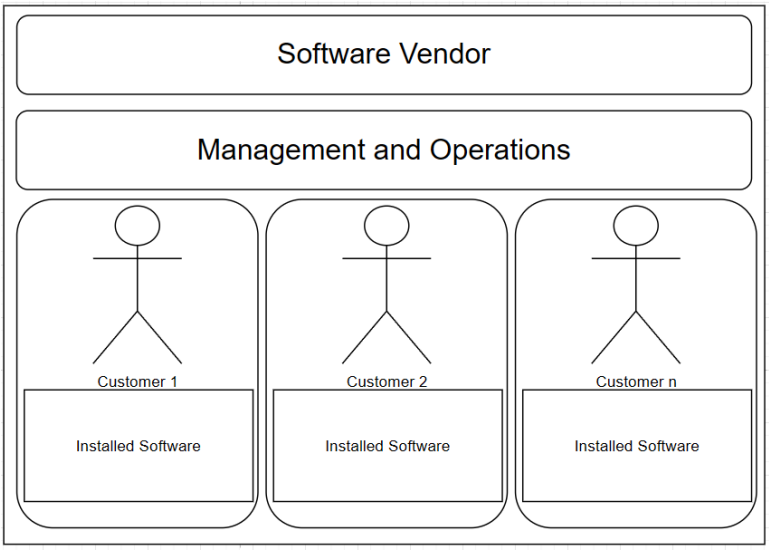
2.1.1 Software as a Service

Software-as-a-service (SaaS) adalah satu dari tiga kategori utama komputasi awan, di samping platform-as-a-service (PaaS) dan infrastructure-as-a-service (IaaS). Ketiga kategori ini memiliki hubungan hierarkis di mana SaaS beroperasi di atas PaaS, yang juga beroperasi di atas IaaS (Tsai et al., 2014). SaaS merupakan sebuah model bisnis dan metode pengantaran perangkat lunak yang memungkinkan perusahaan untuk menyediakan layanan mereka dengan mudah, memfokuskan pada layanan, serta memberikan nilai maksimal baik bagi pelanggan maupun penyedia (Golding, 2024).

Gambar 2.1 dan Gambar 2.2 merupakan ilustrasi model manajemen penyewaan perangkat lunak yang tidak memakai metode SaaS.



Gambar 2.1 Ilustrasi Installed Software Provider

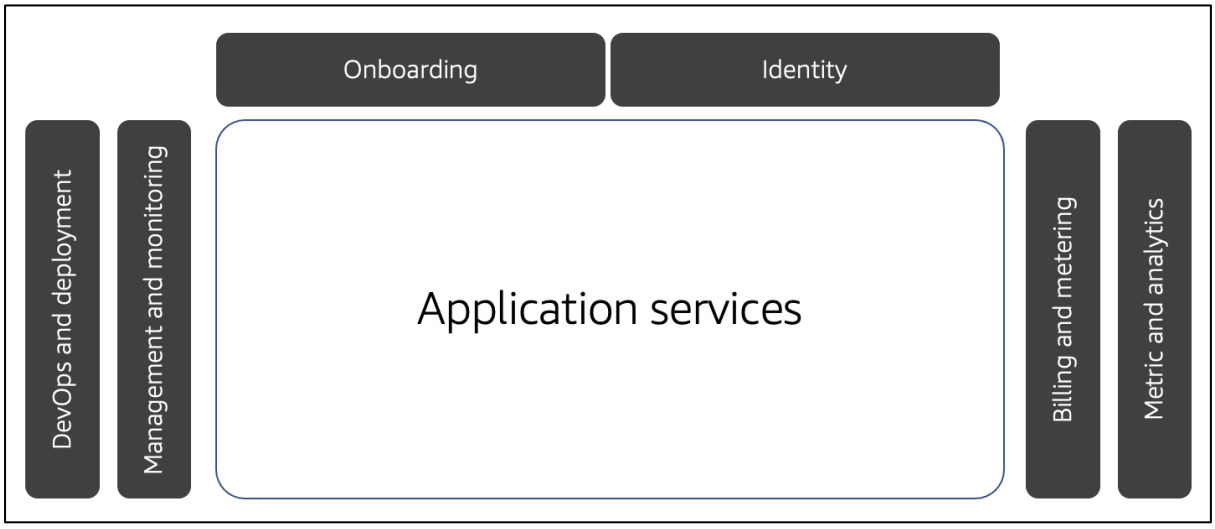


Gambar 2.2 Ilustrasi Managed Service Provider / Application

Installed Service Provider (ISP) adalah model distribusi perangkat lunak yang paling awal sebelum munculnya SaaS. Dalam model ini, perangkat lunak dijual dengan kewajiban bagi pelanggan untuk menginstalnya sendiri. Penjualan ini sering kali disertai dengan layanan profesional tambahan untuk membantu dalam instalasi, kustomisasi, dan konfigurasi sesuai dengan lingkungan masing-masing pelanggan. Namun, model ISP memiliki beberapa kelemahan, seperti memberikan kebebasan kepada setiap pelanggan untuk mengatur lingkungan mereka sendiri, harus mendukung berbagai versi perangkat lunak di antara pelanggan, dan adanya fitur yang khusus dan berbeda untuk setiap pelanggan.

Managed Service Provider/Application Service Provider (MSP/ASP) hadir untuk mengatasi beberapa masalah ini dengan mengonsolidasikan operasional ke satu tim atau entitas pusat. Tim tersebut bertanggung jawab untuk melakukan instalasi, mengatur lingkungan pelanggan, dan menyediakan dukungan, sehingga menghemat sumber daya perusahaan penyedia layanan. Meskipun model MSP mendekati model bisnis SaaS, masih ada perbedaan signifikan, yakni setiap pengguna bisa memiliki versi perangkat lunak yang berbeda. Oleh karena itu, MSP masih memiliki kekurangan dalam hal operasional, karena penyedia layanan harus memiliki tim dukungan khusus yang mampu menangani kebutuhan unik setiap pelanggan.

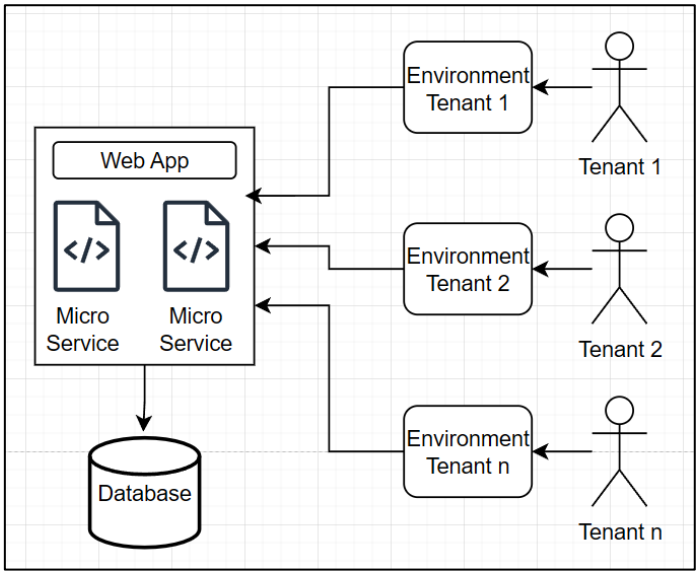
Gambar 2.3 menunjukkan bahwa SaaS muncul sebagai model bisnis yang menyelesaikan masalah-masalah pendahulunya. Penyedia layanan dapat mengatur, mengoperasikan, dan men-*deploy* satu lingkungan aplikasi yang sama untuk semua pengguna melalui "a single pane of glass" (Golding, 2024). Perbedaan fitur di antara pelanggan dapat diakomodasi dengan menyalakan atau mematikan fitur tertentu, sehingga mengurangi beban pengembangan dengan menghilangkan fitur khusus yang disesuaikan untuk setiap pelanggan (Golding, 2024).



Gambar 2.3 Ilustrasi Model Aplikasi SaaS

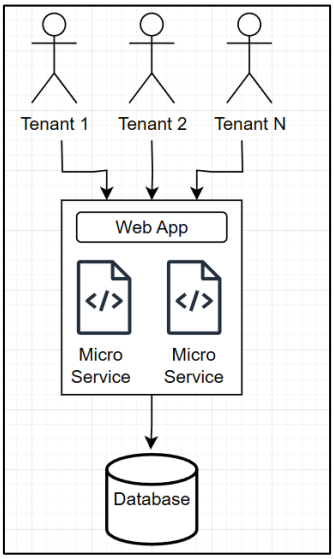
2.1.2 Multi-Tenant Architecture

Gambar 2.4 menggambarkan cara pembuatan sebuah aplikasi dalam environment non-SaaS, di mana aplikasi tersebut biasanya dibangun dengan asumsi bahwa setiap pelanggan akan menginstal dan menjalankannya sendiri. Gambar di bagian kiri menunjukkan ilustrasi sederhana tentang aplikasi yang dibangun dan kemudian dijual kepada pelanggan. Pelanggan memiliki pilihan untuk menginstal aplikasi tersebut di lingkungan mereka sendiri atau menjalankannya di server cloud (Golding, 2024).



Gambar 2.4 Ilustrasi Deployment pada Environment Non-SaaS

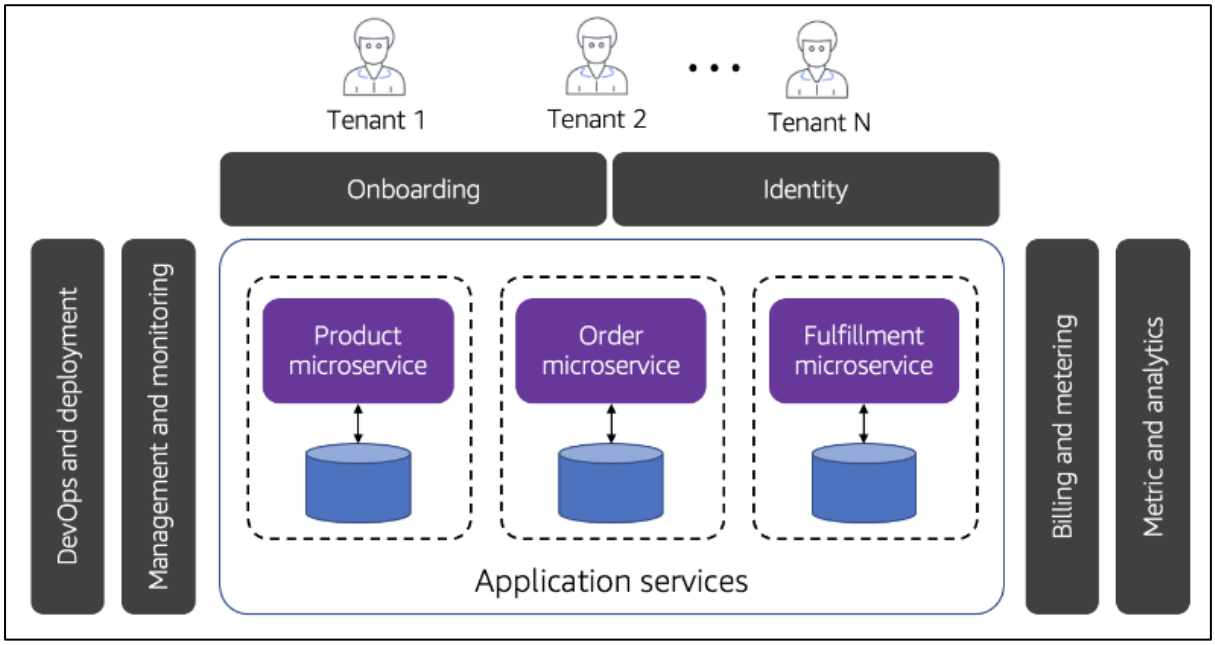
Gambar 2.5 menggambarkan pandangan konseptual dari distribusi aplikasi dalam environment multi-tenant SaaS. Perubahan dalam distribusi ini mempengaruhi cara aplikasi dirancang, dibangun, dan dikelola. Selain itu, istilah yang digunakan untuk pelanggan berubah menjadi tenant, karena penggunaan kata tenant lebih sesuai untuk environment SaaS yang identik dengan model deployment SaaS yang berjalan pada infrastruktur bersama.



Gambar 2.5 Ilustrasi Deployment pada Environment SaaS

2.1.3 Level Kontrol pada Aplikasi SaaS

SaaS bekerja dengan cara memisahkan bagian-bagian yang tidak terkait langsung dengan fungsionalitas aplikasi menjadi dua bagian utama, yaitu Application Plane dan Control Plane. Control Plane, yang sering disebut sebagai "the single pane of glass," berfungsi untuk mengorkestrasi dan mengoperasikan semua komponen yang bergerak dalam aplikasi SaaS. Control Plane juga berfungsi sebagai wadah untuk aplikasi-aplikasi administratif dalam SaaS, seperti Billing, Metric, IAM, Tenant Onboarding dan Management.



Gambar 2.6 Penggambaran Control Plane pada Sistem SaaS

Gambar 2.6 memperlihatkan Control Plane tidak memiliki fungsionalitas untuk mendukung kebutuhan individual tiap tenant, melainkan menyediakan layanan yang merata untuk semua tenant (Golding, 2024). Pendukung kebutuhan tersebut pada Control Plane berupa beberapa komponen, dalam konteks tugas akhir ini adalah kumpulan modul. Deskripsi mengenai nama dan fungsi dari modul-modul tersebut dijelaskan pada Tabel.

Nama	Deskripsi
Onboarding	Mengurus pembuatan tenant dan memicu deployment sumber daya, serta penghubungan komponen-komponen mendasar dalam SaaS kepada tenant seperti identitas user, akun Billing, dan lainnya
Identity Management	Mengurus penghubungan identitas user kepada tenant dan organisasi
Tenant Management	Mengatur sumber daya tenant, dan alur kehidupan tenant pada sistem SaaS
Billing	Mengatur penagihan biaya tenant dan rencana pembayaran tenant

2.1.4 Layanan Manajemen *Instance* Aplikasi SaaS

Instance aplikasi adalah perangkat lunak yang di-*deploy* pada *cloud provider* sebagai sumber daya yang akan dipakai Tenant. *Layanan* manajemen *instance* aplikasi bertugas untuk mengatur jalannya *instance* aplikasi pada *cloud provider*. Layanan manajemen *instance* aplikasi dapat mengatur terjadinya *deployment* atau penghapusan *instance* sesuai dengan aturan bisnis yang ditetapkan pengembang SaaS. Layanan manajemen *instance* aplikasi juga bertugas dalam mendaur ulang *instance* yang sudah ada seperti contohnya bila ada permintaan sumber daya dari Tenant yang bertipe POOL dan *instance* POOL masih ada pada *cloud provider*, layanan tidak akan melakukan *deployment* ulang melainkan mengarahkan Tenant yang membutuhkan sumber daya tersebut ke *deployment* yang sudah ada. Terjadinya *deployment* atau penghapusan pada *cloud provider* dilakukan dengan memodifikasi *instance* aplikasi pada Virtual Machine atau yang akan diterapkan pada tugas akhir ini, pada *cloud run* di *cloud provider* (Zheng & Du, 2014). modul Tenant Management bertindak sebagai layanan manajemen *instance* aplikasi untuk mengatur sumber daya Tenant yang di-*deploy* ke *cloud provider.* Semua aturan bisnis yang mendukung alur kehidupan Tenant ditanamkan pada modul Tenant Management agar aplikasi SaaS dapat berjalan dengan semestinya.

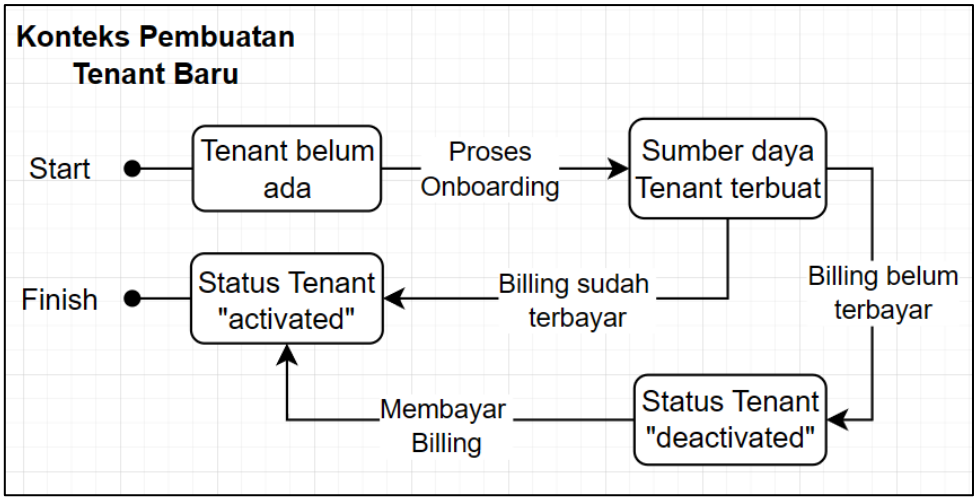
2.2 Arsitektur Kerangka Kerja

Kerangka kerja SaaS merupakan sistem yang mendorong mempercepat pengembangan perangkat lunak bisnis. Dalam konteks SaaS sebagai model bisnis tujuannya adalah untuk meningkatkan daya saing dan kesempatan untung lebih besar. Sebagai kerangka kerja SaaS dengan arsitektur multi-tenan*t*, poin penting yang harus diperhatikan adalah tingginya ketersediaan akses data dan kemampuan untuk diekstensikan, karena sebagai kerangka kerja SaaS harus dapat memudahkan pengembang untuk mengembangkan SaaS yang sesuai dengan proses bisnis pengemebang dengan cepat (Morakos & Meliones, 2014).

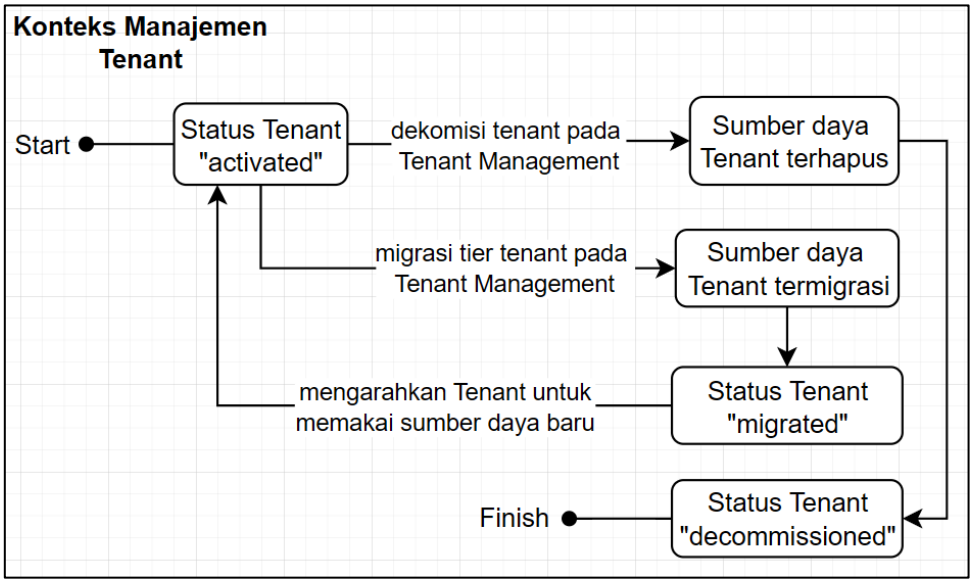
2.3 Tenant Lifecycle

Tenant memiliki beberapa status yang dapat berubah seiring berjalannya proses bisnis, tugas dari manajemen tenant adalah untuk mengatur jalannya perubahan fase tenant dalam sistem. Dalam *lifecycle* tenant pada umumnya, terdapat beberapa fase atau keadaan seperti aktivasi/deaktivasi tenant, penghapusan tenant dan perubahan tingkat langganan. Modul Tenant Management harus dapat memicu perubahan fase serta menangani dampak dari perubahan fase tersebut. Gambar 2.7 menjelaskan keadaan alur kehidupan tenant pada saat proses pembuatan tenant baru. Tenant yang belum terbuat sebelumnya akan dibuat dalam proses *onboarding* yang dilakukan modul Onboarding dengan cara pengguna SaaS memilih produk yang disediakan SaaS. Setelah sumber daya tenant terbuat, status tenant akan berubah menjadi “deactivated” ketika belum membayar Billing. Keadaan alur kehidupan tenant dalam konteks pembuatan tenant baru akan berakhir pada perubahan status tenant menjadi “activated” yang terjadi ketika Billing sudah terbayar. Gambar 2.8 menjelaskan keadaan alur kehidupan tenant dalam konteks manajemen tenant yang sudah ada. Jika melakukan migrasi *tier* tenant pada modul Tenant Management, keadaan status tenant akan menjadi “migrated” pada saat setelah selesai melakukan migrasi. Status tenant akan kembali aktfi setelah tenant telah diarahkan ke sumber daya baru hasil dari migrasi. Karena keadaan tenant kembali aktif, alur kehidupan tenant setelah migrasi tidak akan berakhir, berbeda dari proses dekomisi tenant. Jika melakukan dekomisi tenant pada modul Tenant Management, maka yang terjadi adalah sumber

daya tenant akan terhapus dan status tenant berubah menjadi “decommissioned”. Pada status dekomisi tenant, alur kehidupan tenant akan berakhir karena tenant sudah terhapus.



Gambar 2.7 State Diagram Lifecycle Pembuatan Tenant Baru



Gambar 2.8 State Diagram Lifecycle Manajemen Tenant

2.2.1 Aktivasi Tenant

Setelah registrasi tenant baru, proses yang diinginkan dari tenant setelahnya tentu saja bagaimana cara mendapatkan akses layanan dari sistem, dengan ini tenant harus masuk ke dalam fase aktivasi dan akan memicu proses *onboarding* yang akan mengonfigurasi komponen infrastruktur yang akan dipakai oleh tenant baru (Golding, 2024).

2.2.2 Deaktivasi Tenant

Saat tenant ingin melakukan deaktivasi, manajemen tenant memiliki tanggung jawab untuk mengatur kebijakan di mana tenant akan diblok dalam menggunakan layanan sistem. Aksi ini juga dapat terhubung kepada modul *Identity Management* yang akan menonaktifkan semua user pada tenant dalam memakai layanan sistem. Manajemen tenant fase ini juga dapat terhubung kepada modul Billing di mana ketika tenant berhenti melakukan pembayaran, modul

Billing akan mengidentifikasi tenant tersebut dan memicu status yang akan membuat tenant masuk ke dalam fase deaktivasi (Golding, 2024).

2.2.3 Penghapusan Tenant

Berbeda dari fase deaktivasi di mana tenant yang tidak aktif dapat secara simple aktivasi ulang dan dapat melanjutkan memakai layanan sistem, penghapusan tenant berarti menghapus konfigurasi infrastruktur dan atau segala sumber daya yang dipakai oleh tenant. Mengapa fase ini mungkin diperlukan? Ada berbagai macam factor, salah satunya jika tenant sudah terlalu lama non-aktif data-data dan konfigurasi tenant yang tidak diperlukan akan berkontribusi kepada kompleksitas proses yang tidak menambah nilai apa-apa terhadap bisnis (Golding, 2024). Ada beberapa strategi dalam menghapus tenant:

1. Menghapus sumber daya yang berhubungan kepada tenant
2. Menghapus secara komplit data tenant dari sistem
3. Mengarsip data tenant sebelum menghapus dari sistem

Mengarsip data tenant bertujuan agar dapat menghidupkan kembali tenant yang telah dihapus dengan mudah tanpa harus menambah kompleksitas pada sistem. Hal ini bisa dilakukan dengan cara memindahkan unsur-unsur dari tenant ke pada tempat yang memiliki impak serta kompleksitas kecil terhadap sistem (Golding, 2024).

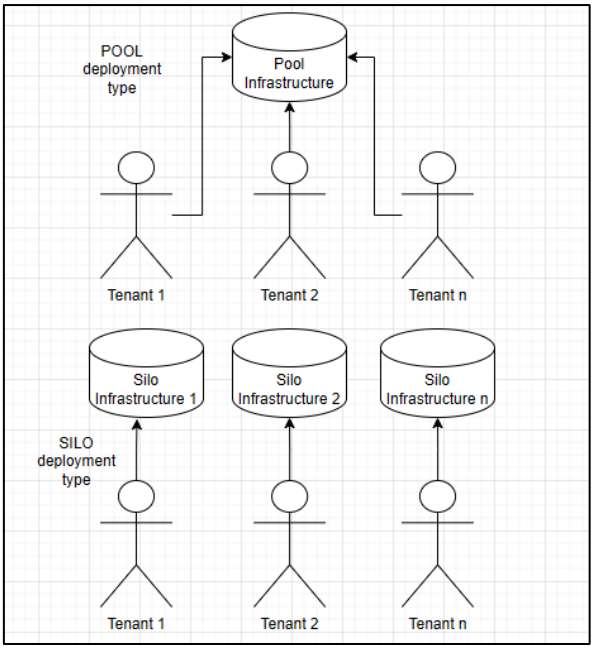
2.2.4 Perubahan *Tier* Langganan

Tier atau tingkatan langganan secara singkat mengatur tentang layanan apa saja yang didapatkan oleh tenant. Kompleksitas *tiering* langganan mengikuti lingkungan kebijakan dan kebutuhan yang diterapkan oleh SaaS itu sendiri. Pada kompleksitas rendah, *tiering* tidak menimbulkan banyak komponen yang bergerak dalam sistem dan umumnya hanya mengubah satu bagian saja seperti mengubah kebijakan *throttling* sumber daya yang akan digunakan tenant. Pada model yang lebih kompleks lagi di mana perubahan *tier* langganan akan menyebabkan banyak komponen yang harus dimigrasi seperti memindahkan *stack* tenant yang sebelumnya ada di dalam *POOL deployement* tingkat dasar ke *POOL deployment* tingkat premium yang memiliki spesifikasi berbeda**.** Hal ini akan bertambah rumit jika SaaS memiliki kebutuhan *zero downtime deployment* (Golding, 2024).

2.4 Model Deployment Tenant

Aplikasi SaaS multi-tenant memungkinkan banyak tenant untuk menggunakan layanan. Tenant dapat memilih layanan berdasarkan *tier* yang mempengaruhi model deployment SaaS yang akan diterapkan. Ada dua model deployment dalam aplikasi SaaS: SILO dan POOL (Amazon Web Services, 2024) yang diilustrasikan pada Gambar 2.9.

Dalam model SILO, sumber daya disediakan eksklusif untuk satu tenant. Sebaliknya, model POOL memungkinkan sumber daya digunakan oleh banyak tenant. Kedua model ini dapat diterapkan pada seluruh atau sebagian sumber daya tenant. Pemilihan model deployment memengaruhi skalabilitas, ketersediaan, isolasi data, penggunaan sumber daya, routing, kompleksitas, dan biaya yang ditanggung oleh penyedia SaaS (Golding, 2024).



Gambar 2.9 Metode Deployment Infrastruktur Tenant pada SaaS Multi-Tenant

2.5 Tantangan Arsitektural

Setiap desain arsitektur memiliki kelebihan dan kekurangan, dan setiap kekurangan merupakan tantangan yang harus diatasi agar tidak mengurangi performa dan juga keuntungan. Arsitektur multi-tenant juga tidak lepas dari hal tersebut, pada bagian ini akan membahas apa saja hal yang harus diperhatikan saat mengembangkan modul Tenant Management dalam aplikasi SaaS.

2.5.1 Performa

Multi-tenant saling berbagi sumber daya dalam aplikasi, hal yang tidak diinginkan adalah satu tenant memakan dan menyumbat semua sumber daya pada aplikasi yang dapat berefek kepada tenant lainnya. Salah satu metode untuk menangani hal tersebut adalah dengan memberikan jumlah sumber daya yang sama pada semua *instance* yang divirtualisasi, namun metode ini memiliki potensi untuk menimbulkan inefisiensitas dalam utilisasi sumber daya (Bezemer & Zaidman, 2010). Solusi lainnya adalah dengan mengisolasi tenant yang agresif dalam menggunakan sumber daya sehingga utilisasi sumber daya tenant tersebut dibatasi agar tidak mencapai level berbahaya (Li et al., 2008).

2.5.2 Keamanan

Lingkungan multi-tenant ada dalam satu aplikasi dan basis data yang sama, kegagalan dalam menjaga keamanan tenant dapat menimbulkan terbongkarnya data seluruh tenant yang ada. Masalah keamanan sangat penting dan harus menjadi salah satu prioritas, maka dari itu diperlukan isolasi keamanan. Ketika tenant ingin mengintegrasikan layanan dari SaaS ke domain pribadi mereka, harus ada pencegahan kepada user tertentu yang terkait pada tenant untuk dapat mengakses tenant lainnya. Hal ini dapat dicapai dengan memberikan id unik kepada tenant, dan cabang *directory* yang diisolasikan dengan tenant id pada proses *onboarding*. Lalu admin tenant dapat dengan mudah memetakan user yang akan masuk ke dalam domain pribadi tenant (Guo et al., 2007).

Aspek isolasi keamanan lainnya adalah isolasi kontrol akses, yang berarti mencegah pengguna untuk mendapat hak dalam mengakses sumber daya punya tenant lain. Strategi ini dapat dicapai dengan filter implisit dan izin eksplisit. Menggunakan filter implisit berarti menentukan izin akses dengan memilah jenis sumber daya apa yang diminta, serta peraturan aksesnya terhadap atribut tertentu dalam tenant, atribut ini bisa apa saja yang secara umum adalah id unik tenant. Contohnya adalah perintah SQL seperti "where tenant_id = 'foo'". Selanjutnya izin eksplisit mengarah kepada izin akses yang sebelumnya sudah didefinisikan kepada tenant yang bersangkutan. Maka dari itu tidak perlu adanya penambahan delegasi umum pada seluruh tenant (Guo et al., 2007).

Tujuan utama dari isolasi keamanan adalah melindungi data tenant, hal tersebut secara praktis dapat dilakukan menggunakan isolasi informasi. Informasi yang disimpan dalam basis data dan di transfer melalui jaringan secara tradisional dilindungi dengan sistem enkripsi dan *digital signature*, namun dalam sistem multi-tenant menggunakan kunci enkripsi yang sama pada setiap tenant membuat metode tersebut menjadi tak berarti, karena hanya dapat melindungi informasi dari serangan eksternal (diluar sistem tenant). Dalam kasus ini setiap tenant harus punya kunci enkripsinya masing-masing (Guo et al., 2007).

2.6 Cloud

Aplikasi multi-tenant biasanya merupakan layanan berbasis *cloud* dengan menggunakan satu *instance* aplikasi yang berbagi *hardware*, infrastruktur, *basis data* dan virtualisasi. Dengan berbagi sumber daya *hardware*, dapat meningkatkan utilitas hardware yang menghasilkan biaya keseluruhan lebih murah. Pada infrastruktur multi-tenant dalam *cloud*, dapat secara mudah menambah daya pada aplikasi dengan cara menambah atau mengalokasikan hardware, hal ini menjadikan semua tenant *scalable* karena menggunakan infrastruktur dan *basis data* yang sama. Salah satu keuntungan menggunakan *cloud* lainnya adalah jaminan ketersediaan datanya dan dapat diakses dari mana saja (Kulkarni et al., 2013).

Dalam rancang bangun sistem manajement tenant untuk aplikasi SaaS ini akan menggunakan Google Cloud Computing (GCP) karena menyediakan banyak layanan yang akan dibutuhkan pada pengembangan kerangka kerja tugas akhir ini seperti Cloud Run yang akan digunakan untuk mendeploy instance aplikasi produk, Cloud SQL sebagai basis data produk dan Pub/Sub yang menjadi distributor *event* yang akan dipublish oleh setiap modul dalam SaaS sebagai sarana komunikasi asinkron. Selain itu GCP juga menyediakan support *runtime* bahasa pemrograman Go yang akan kita pakai pada rancang bangun ini. GCP menggunakan metode Billing *pay as you go, on-demand price* yang membuat biaya hosting menjadi lebih fleksibel berdasarkan kebutuhan pemakaian (Wankhede et al., 2020).

2.7 Bahasa Pemrograman Go

Go merupakan bahasa pemrogramman yang dikembangkan oleh Google untuk menangani layanan *backend*. Go dibangun dari level rendah, yang membuat Go memiliki performa yang tinggi, selain itu Go juga menyediakan *Garbage Collector*, membuat Go aman dari isu *memory leak* (Andrawos & Helmich, 2017). Go adalah *compiled language* yang berarti hasil program Go akan diubah menjadi *machine code,* hal ini meningkatkan kecepatan Go karena program sudah *dicompile* terlebih dahulu. Go juga merupakan *static typed language* yang membuat pergerakan tipe data konsisten, mudah dibaca dan membuat program lebih cepat karena tidak

perlu mengecek tipe data saat *runtime* (Donovan & Kernighan, 2015). Salah satu keunggulan Go lainnya adalah dapat menggunakan konsep pararelsime dalam program dengan memakai *goroutine*, hal ini memungkinkan sebuah proses berjalan pararel dan tidak menghambat proses lainnya yang lebih dibutuhkan untuk berjalan.

Go dikembangkan di era *cloud computing,* dan sudah mempertimbangkan teknologi perangkat lunak modern, selain itu Go teroptimisasi untuk arsitektur *microservice* karena program go *dicompile* menjadi satu file *binary,* menjadikan Go tidak memerlukan *virtual machine* di lingkungan *production*. Hal ini cocok untuk menggunakan Go dalam mengembangkan perangkat lunak dalam *cloud* yang akan diterapkan pada tugas akhir ini (Andrawos & Helmich, 2017).

2.8 Kerangka Kerja Next.js

Next.js merupakan kerangka kerja *fullstack* berbasis React. Next.js menyediakan fitur-fitur seperti routing berbasis *file-system, client side / server side rendering,* pengambilan data menggunakan *fetch* untuk API, optimisasi gambar, *font* dan *script*; Pengembangan *frontend* dalam Next.js menggunakan *TypeScript* yang telah ditingkatkan untuk mendapatkan kompilasi dan pengecekan tipe variabel lebih baik (Vercel Inc., n.d.). Next.js cocok digunakan dalam pengembangan *frontend* tugas akhir ini karena akan memanfaatkan fitur *fetching* API, dan optimisasi gambar fonr dan script.

2.9 Terraform

Terraform adalah *platform* infrastruktur yang memungkinkan pengembang untuk mendefinisikan *cloud* dan sumber daya lokal yang dimiliki pengembang hanya dengan file konfigurasi yang dapat mudah dibaca. File konfigurasi yang ditulis pada Terraform dapat digunakan ulang, dibagikan, dan dipecah menjadi beberapa versi yang memudahkan pengembang dalam memanajemen infrastruktur pada siklus hidup aplikasi. Terraform dapat mengatur komponen level rendah seperti komputasi, penyimpanan, dan sumber daya *networking*; juga mengatur komponen level tinggi seperti DNS dan fitur SaaS (HashiCorp, n.d.). Terraform akan digunakan sebagai otomasi *deployment* aplikasi tenant pada tugas akhir ini.

2.10 Event Driven Architecture

Event Driven Architecture (EDA) merupakan desain fundamental dari manajemen *state* pada aplikasi secara asinkron. Pemrosesan event yang dilakukan secara asinkron berdampak positif bagi komunikasi antar aplikasi/modul yang akan didistribusikan secara besar, serta membuat aplikasi/modul tersebut menjadi tidak terlalu terikat satu dengan yang lainnya (Stack, 2022). Pengurangan dependensi antar modul akan menjadikan sebuah modul dapat berdiri sendiri tanpa membutuhkan modul spesifik, hal ini dibutuhkan dalam konteks tugas akhir ini yang bertujuan merancang kerangka kerja modul Tenant Management, karena pengembang yang akan menggunakan modul ini dapat mengintegrasikan modul lainnya secara mudah karena tidak adanya dependensi langsung antar modul.

EDA sangat cocok untuk digunakan dalam pengembangan tugas akhir ini menggunakan bahasa Go, karena Go yang menyediakan *goroutine* untuk memroses *event* secara asinkron dan

melakukan proses bisnis berat seperti *deployment* sumber daya ke GCP dibalik agar tidak menghambat proses bisnis tenant.

2.11 Domain Driven Design

Domain Driven Design (DDD) adalah pola pengembangan perangkat lunak yang berorientasi pada keselarasan antara desain perangkat lunak dan domain bisnis. Tujuan utamanya adalah mengatasi kegagalan proyek perangkat lunak dengan meningkatkan komunikasi antara pengembang dan ahli bisnis. Pola DDD biasa digunakan dalam desain arsitektur mikroservis karena penekanan pemahaman mendalam atas domain bisnis untuk memastikan aplikasi dapat mempertahankan model bisnis yang konsisten dan juga pemisahan komponen yang jelas untuk mendapatkan kemampuan integrasi yang baik. Maka dari itu pola DDD cocok untuk digunakan pada pengembangan aplikasi tugas akhir ini.

DDD terbagi menjadi dua bagian utama: strategis dan taktis. Bagian strategis digunakan untuk menganalisis domain bisnis dan strategi untuk memuaskan domain bisnis tersebut, serta mendekomposisi sistem menjadi komponen-komponen yang terintegrasi dengan baik. Sementara itu, bagian taktis memungkinkan pengembang menulis kode yang mencerminkan domain bisnis, mencapai tujuan bisnis, dan menggunakan bahasa bisnis yang tepat. Dengan pola ini, pengembangan perangkat lunak menjadi lebih efektif dan efisien karena DDD membantu pengembang dalam memahami domain bisnis untuk mengarahkan keputusan desain sesuai dengan strategi bisnis (Khononov, 2021).

2.12 Command Query Responsibility Segregation

Berbeda dari arsitektur MVC (Model View Controller) di mana request dari client diterima controller yang memakai satu komponen "model" untuk melakukan tanggung jawab *read* dan *write* data ke basis data, arsitektur CQRS (Command Query Responsibility Segregation) memisahkan tanggung jawab *read* dan *write* menjadi dua komponen, *Command Model* dan *Query Model*. Arsitektur CQRS mengikuti pedoman *Single Responsibility* oleh (Martin, 2006), karena dengan memisahkan tanggung jawab developer dapat menunjuk dengan tepat komponen mana yang menyebabkan masalah.

Menurut Erb & Kargl, 2014 (dalam Korkmaz & Nilsson, 2014) pemisahan tanggung jawab model juga menambah fleksibilitas dan kepastian dalam manajemen data. seperti pada kasus *Command* yang memiliki tanggung jawab *write* diarahkan ke basis data *master* dan *Query* yang memiliki tanggung jawab *read* diarahkan ke basis data *slave,* dengan memisahkan pemakaian basis data menurut Korkmaz & Nilsson, 2014 (dalam Rajković et al., 2013) akan menaikkan waktu *response* karena tanggung jawab basis data *master* yang melakukan *write* tidak akan terhambat oleh permintaan *read* dan begitu juga sebaliknya untuk basis data *slave*. Untuk karena itu sistem segregasi tanggung jawab CQRS cocok digunakan pada tugas akhir ini sebagai kerangka kerja.

2.13 Komunikasi Antar-Modul

Agar bisa menjadi aplikasi SaaS secara keseluruhan, tiap modul akan saling berkomunikasi untuk mengerjakan tugas yang memiliki hubungan dengan modul lain di domain nya masing-masing. Mayoritas modul dalam SaaS akan membutuhkan tenant *identifier* jika mengerjakan

proses yang berhubungan dengan *lifecycle* tenant. Dengan adanya id tenant resiko untuk collision dan kesalahan identitas saat memproses tenant akan berkurang, maka dari itu sumber kebenaran tenant harus berasal dari modul Tenant Management. Berikut akan membahas proses hubungan modul Tenant Management terhadap modul-modul lain.

2.13.1 Komunikasi Terhadap Billing

Modul Tenant Management berkomunikasi dengan modul Billing untuk mendapatkan data *tier* produk yang dideskripsikan pada **Tabel 2.*1*** dan pada saat perubahan *tier* langganan yang dideskripsikan pada **Tabel 2.*2***. Modul Billing mempunyai proses internal yang diurus secara mandiri oleh modul Billing seperti merubah nilai biaya yang ditagihkan kepada tenant.

Tabel 2.1 Deskripsi Endpoint API Billing Mendapatkan Daftar *Tier* Produk

API Endpoint	**Method**	**Request Header**	**Keterangan**
/api/v1/products/	GET	-	Endpoint API untuk mendapatkan daftar produk yang tersedia

Contoh Json Response:	{
	"data": [
	{
	"id": "8a5f8dc1-b751-4bf1-ad97-42092b50c80a",
	"app_id": "4",
	"name": "SaaS Todos V2",
	"tier_name": "Saas Todo V2 Basic",
	"tier_index": 0,
	"price": [
	{
	"id": "8a5f8dc1-b751-4bf1-ad97-42092b50c80a",
	"product_id": "",
	"price": 100000,
	"reccurence": "monthly"
	}
	]
	}
	]
	}		

Tabel 2.2 Deskripsi Endpoint API Billing Membuat Tagihan untuk Tenant

API Endpoint	**Method**	**Request Header**	**Keterangan**
/v1/jwt/tenants	POST	Authorization: Bearer {jwt}	Endpoint API untuk mendapatkan link pembayaran

Contoh Request Body:	{
	"data": {
	"price_id": "0ff63e16-5724-4a37-a835-9320a7869e4b",
	"org_id": "c16d7bc1-a3db-46b7-9a7f-1bdde743909c",
	"tenant_id": "29d8ca39-2633-4a6d-8121-13a7e6577dc4",
	"tenant_name": "testing"
	}
	}

Contoh Json Response:	{
	"data": {
	"billing_url": "example.com"
	},
	"message": "success"
	}

2.13.2 Komunikasi Terhadap Proses Onboarding

Saat tenant pertama kali melakukan registrasi, modul Onboarding akan melakukan *request* kepada modul Tenant Management untuk membuatkan *identifier* pada tenant tersebut, dan menyimpan data-data registrasi tenant. *Identifier* tersebut nantinya akan dipakai oleh modul Onboarding pada proses internalnya dalam mengonfigurasi infrastruktur tenant barunya.

2.13.3 Komunikasi Terhadap *Identity Management*

Modul Tenant Management berkomunikasi dengan modul *Identity Management* untuk mendapatkan Kumpulan data organisasi yang dimiliki oleh pengguna aplikasi SaaS yang dideskripsikan pada **Tabel 2.*3***. Pengguna aplikasi SaaS dapat memilih organisasi mana yang menjadi target untuk mengurus kumpulan tenant pada organisasi tersebut.

Tabel 2.3 Deskripsi Endpoint API Mendapatkan Daftar Organisasi yang Dimiliki Pengguna

API Endpoint	Method	Request Header	Keterangan
/api/organization	GET	Authorization: Bearer {jwt}	Endpoint API untuk mendapatkan daftar organisasi yang dimiliki pengguna aplikasi SaaS

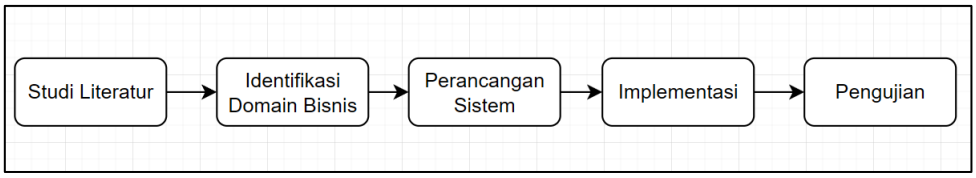
Contoh Json Response:	{
	"code": int,
	"data": [
	{
	"organization_id": string,
	"name": string
	}
	]
	}		

<halaman ini sengaja dikosongkan>

BAB 3 METODOLOGI

3.1 Metode yang Dirancang

Tugas akhir ini dibangun menggunakan metode yang bertindak sebagai panduan agar pengembang memiliki dasar metode yang kuat dalam mengembangkan kerangka kerja aplikasi SaaS multi-tenant. Metode-metode yang digunakan diilustrasikan dalam Gambar 3.1 dan dijabarkan pada sub-bab 3.1.1 sampai 3.1.5:



Gambar 3.1 Metode yang Dirancang

3.1.1 Studi Literatur

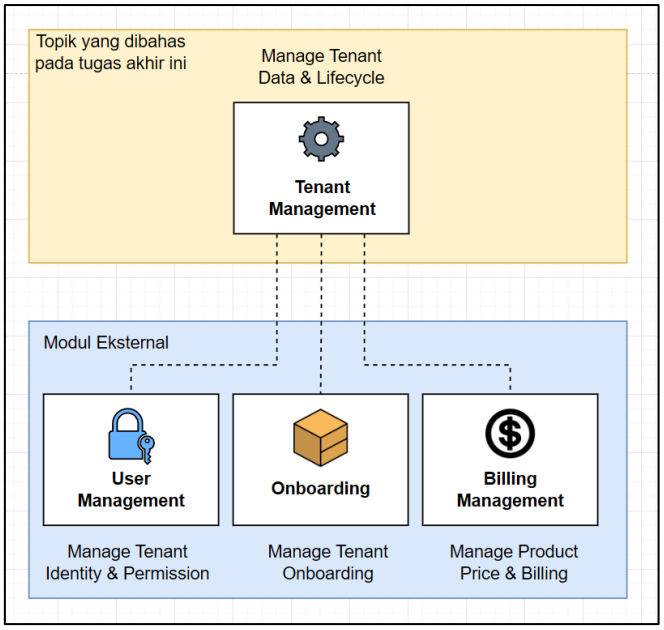
Pada tahap ini dilakukan studi literatur dari referensi seperti jurnal ilmiah dan buku, hal ini bertujuan untuk memahami materi-materi yang dibutuhkan dalam pengerjaan tugas akhir. Berikut materi-materi yang dibutuhkan:

1. Fundamental Tenant Management (sub-bab 2.3)
2. *Cloud Computing* (sub-bab 2.5)
3. *Event Driven Architecture* (sub-bab 2.9)
4. *Domain Driven Design* (sub-bab 2.10)
5. *Command Query Responsibility Segregation* (sub-bab 2.11)

3.1.2 Identifikasi Komponen Domain

Di tahap berikut dilakukan identifikasi komponen dengan merancang domain dari tiap modul yang ada dalam *control plane*. Domain yang dimaksud adalah tanggung jawab masin-masin komponen atau dalam konteks tugas akhir ini dipecah menjadi modul. Setiap modul dapat memengaruhi satu sama lain secara tidak langsung tanpa melanggar cakupan tugas atau domain yang telah diidentifikasi. Perlu dilakukan pemetaan relasi domain tiap komponen agar jelas dalam menentukan strategi bisnis yang tepat dalam membangun aplikasi SaaS.

Gambar 3.2 merupakan gambaran secara garis besar cakupan pengaruh tiap modul secara garis besar. Modul *Onboarding* merupakan titik awal pelanggan (user aplikasi SaaS) untuk dapat memulai menyewa sumber daya dan menjadikan dirinya sebuah tenant. Modul Tenant Onboarding akan mengatur terjadinya pembuatan sumber daya tenant, serta komponen-komponen lain yang akan dipunyai tenant seperti akun pembayaran (Alexander, 2024); Modul Tenant Management dibahas pada topik tugas akhir ini, bertugas dalam mengatur keluar masuknya data tenant pada aplikasi SaaS serta mengurus perubahan *lifecycle* tenant. Perubahan *lifecycle* tenant menyangkut sumber daya yang dimiliki tenant itu juga. Sumber daya tenant akan dapat dimigrasi atau dihapus oleh modul Tenant Management sesuai dengan kebutuhan bisnis; Modul Billing bertugas dalam penyimpanan data produk, harganya *tier* produk, serta melakukan penagihan pembayaran kepada tenant (Rachmat, 2024); Modul *Identity Access Management* bertugas dalam mengurus autentikasi dari pelanggan. modul IAM juga mengatur hak akses tenant pada sebuah sumber daya tertentu (Hilmi, 2024).

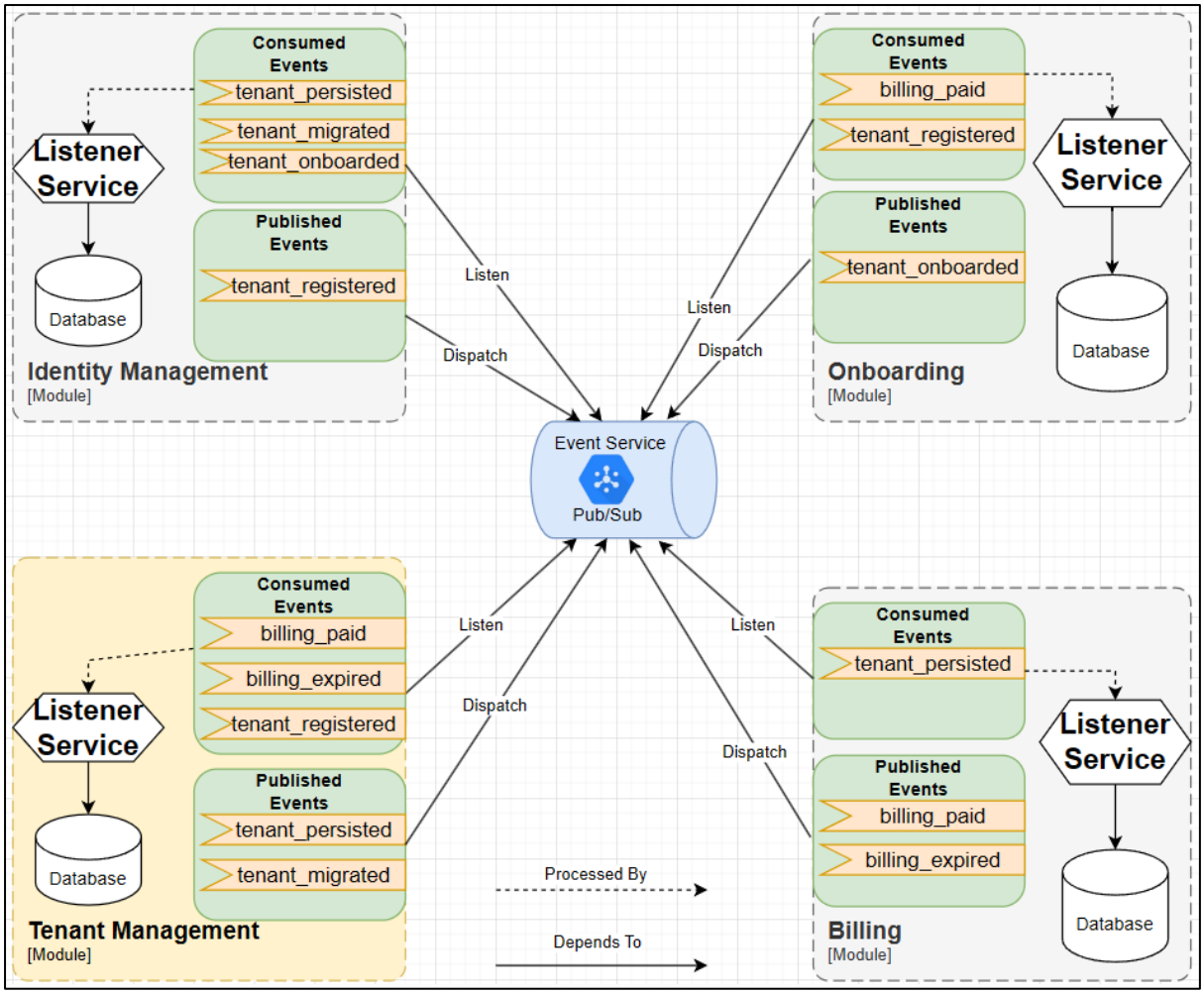


Gambar 3.2 Relasi Antar Modul Keseluruhan

Tugas akhir ini akan merancang sekaligus membangun kerangka kerja modul Tenant Management. Perlu dilakukannya identifikasi komponen domain terhadap relasi modul Tenant Management dengan modul-modul lainnya agar kebutuhan bisnis dapat terpenuhi secara efisien. Dengan adanya identifikasi komponen domain, pengembangan modul Tenant Management hanya perlu fokus dalam memahami konteks yang dibutuhkan modul Tenant Management saja tanpa mengalihkan perhatian pengembangan yang dapat menyebabkan inefisiensi dari segi waktu dan usaha. Penjabaran komponen domain relasi modul Tenant Management dengan modul-modul lainnya dijabarkan sebagai berikut:

1. Modul Tenant Management akan mengurus tenant. Modul-modul lainnya akan berelasi dengan modul Tenant Management dalam konteks proses bisnis yang memerlukan data tenant
2. Semua proses bisnis di modul lain yang memerlukan perubahan *lifecycle* tenant, akan diurus pada modul Tenant Management, maka dari itu modul yang membutuhkan perubahan *lifecycle* tenant akan berelasi dengan modul Tenant Management.
3. Modul Tenant Management akan berelasi dengan modul *Identity Management* dalam konteks proses bisnis yang memerlukan data pelanggan (user aplikasi) dalam Tenant.
4. Modul Tenant Management akan berelasi dengan modul Billing dalam konteks proses bisnis yang berhubungan dengan data produk, harga produk, serta penagihan pembayaran tenant.

Melakukan relasi antar modul akan dilakukan menggunakan metode Event Driven Design, karena itu pada tugas akhir ini perlu dilakukan pemetaan *event* untuk mengetahui kebutuhan konsumsi event per modul dan sumber yang menghasilkan event tersebut. Hasil dari event sourcing tersebut merupakan daftar kumpulan event yang akan di konsumsi dan di terbitkan dari modul-modul dalam sistem aplikasi SaaS (digambarkan pada Gambar 3.3).



Gambar 3.3 Event Source Diagram

3.1.3 Perancangan Sistem

Untuk mengimplementasi perangkat lunak kerangka kerja modul Tenant Management, perlu dilakukan analisis kebutuhan. Detail kasus penggunaan pelanggan (user aplikasi SaaS) dalam konteks dekomisi tenant dapat dilihat pada UC 01 dalam **Tabel 3.1**. Dekomisi tenant merupakan proses untuk menghapus tenant serta sumber daya nya, ketika pelanggan ingin berhenti berlangganan tenant.

Tabel 3.1 Spesifikasi Usecase Dekomisi Tenant

Kode *Use Case*	**UC 01**	
Nama *Use Case*	Dekomisi Tenant	
Aktor	Pengguna aplikasi SaaS	
Deskripsi	*Use case* ini menggambarkan proses mendekomisi tenant yang akan menghapus sumber daya tenant	
Kondisi Awal	Pengguna telah terdaftar, memiliki organisasi dan tenant	
Kondisi Akhir	Tenant terhapus dari sistem	
ALUR NORMAL	Aktor	Sistem Tenant Management
	1. Pengguna memilih organisasi yang akan digunakan untuk memilih tenant	1. Sistem menampilkan daftar tenant dalam organisasi

	2. Pengguna menekan tombol "decommission" pada tenant	2. Sistem menghapus data tenant
		3. Sistem menghapus sumber daya tenant pada *cloud provider*
		4. Sistem menerbitkan *event* bahwa tenant telah terdekomisi untuk didengar modul lain

Konteks kebutuhan fungsional lainnya adalah mengubah *tier* tenant dapat dilihat pada UC 02 dalam **Tabel 3.*2***. Sebuah produk sebelum menjadi tenant memiliki level layanan yang beragam, perbedaan level layanan ini yang disebut dengan *tier.* Mengubah *tier* merupakan proses ketika pelanggan tenant ingin mengubah level layanan tenant, yang biasanya memiliki tetapi tidak terlepas dari perbedaan tipe *deployment* (contoh SILO dan POOL).

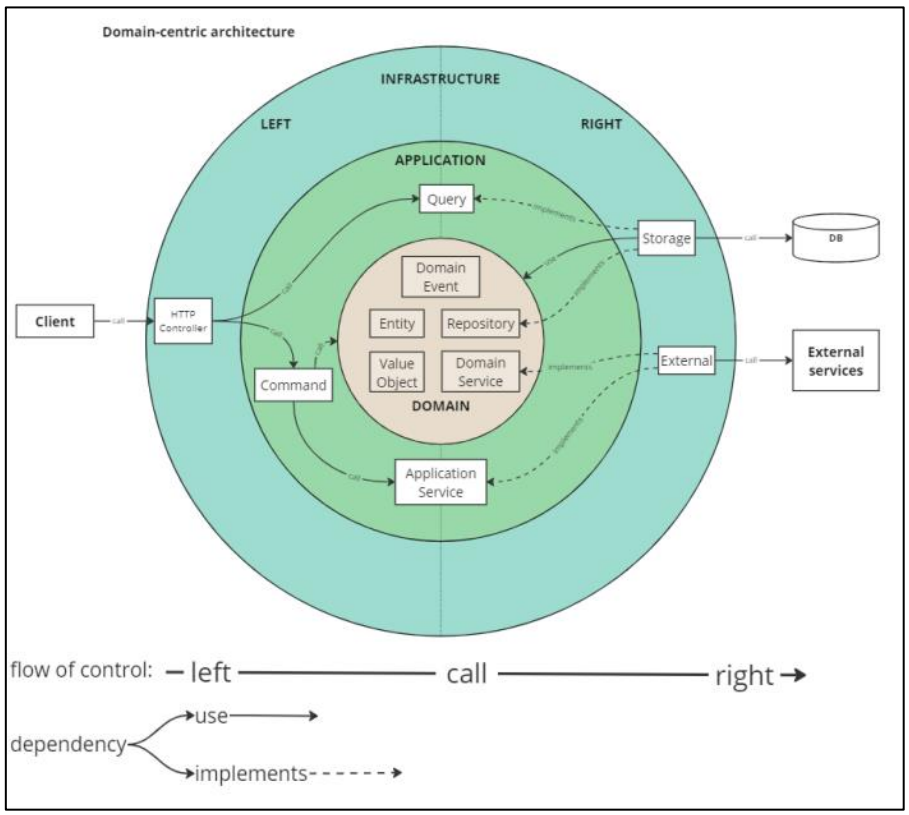
Tabel 3.2 Spesifikasi Usecase Pengguna Mengubah *Tier* Tenant

Kode *Use Case*	**UC 02**	
Nama *Use Case*	Mengubah *Tier* Tenant	
Aktor	Pengguna aplikasi SaaS	
Deskripsi	*Use case* ini menggambarkan proses pemindahan sumber daya Tenant jika adanya perubahan *tier* tenant	
Kondisi Awal	Pengguna telah terdaftar, memiliki organisasi dan tenant	
Kondisi Akhir	Status tenant pengguna berubah menjadi "dalam masa perpindahan *tier*"	
ALUR NORMAL	Aktor	Sistem Tenant Management
	1. Pengguna memilih organisasi yang akan digunakan untuk memilih tenant	1. Sistem menampilkan daftar tenant dalam organisasi
	2. Pengguna memilih tenant pada organisasi	2. Sistem menampilkan informasi detail tenant
	3. Pengguna memilih *tier* pada tenant yang akan diubah	3. Sistem mengubah status tenant menjadi "tenant_migrating"
		4. Sistem berkomunikasi dengan modul Billing untuk mendapatkan link pembayaran
		5. Link pembayaran diberikan kepada pengguna untuk dibayar secara asinkron

3.1.4 Implementasi Perangkat Lunak

Tugas akhir ini akan menggunakan Arsitektur CQRS pada pola DDD yang skemanya diilustrasikan pada Gambar 3.4. DDD digunakan agar dapat menentukan strategi yang efisien serta memastikan model bisnis modul Tenant Management tetap konsisten saat melakukan integrasi dengan modul-modul lainnya. Dalam pola DDD terdapat 2 sisi: sisi kiri dan sisi kanan, di mana alur kontrolnya dari *request client* mulai dari bagian kiri, dan akan mengalir ke komponen-komponen ke arah kanan. DDD mempunyai 3 layer tanggung jawab: "infrastructure" yang berhubungan langsung dengan basis data dan servis eksternal;

“application” merupakan antarmuka yang akan dipanggil oleh *controller* di sisi kiri untuk keperluan proses bisnis; “domain” adalah inti dari DDD, yang menampung abstraksi yang merupakan hasil dari strategi dan taktik bisnis.



Gambar 3.4 Alur Arsitektur CQRS pada DDD

3.1.5 Pengujian dan Evaluasi Hasil

Pengujuan dilakukan menggunakan *functional* testing yang bertujuan untuk memastikan mekanisme dari layanan-layanan yang ada di dalam modul berkerja sesuai kebutuhan.

3.2 Peralatan Pendukung

Dalam pengerjaan tugas akhir ini, penulis menggunakan alat-alat sebagai berikut:

1. VSCode
2. Github
3. GCP
4. Terraform
5. Draw.io

VsCode akan digunakan sebagai code editor selama pengembangan. Github dipakai sebagai *repository* yang akan menyimpan sumber kode hasil pengembangan. GCP merupakan *cloud provider* yang menjadi tempat untuk *deployment instance* aplikasi, dan sebagai penyedia layanan *event* untuk berkomunikasi antar modul. Terraform adalah alat yang dikembangkan oleh perusahaan Hashicorp sebagai alat pembantu untuk melakukan *deployment* ke *cloud provider*. Draw.io dipakai untuk membuat gambar dan diagram yang menjelaskan alur dan komponen-komponen pada tugas akhir ini.

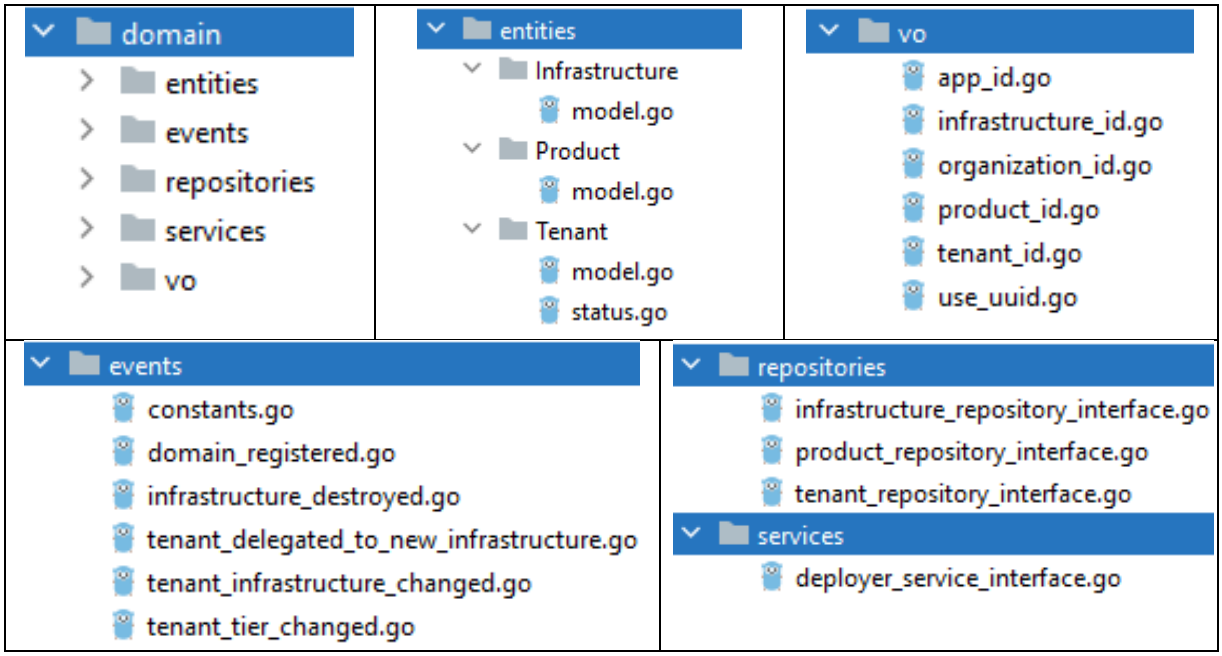
3.3 Implementasi

3.3.1 Struktur Kode Sumber

Kerangka kerja dirancang dan dibangun menggunakan arsitektur CQRS pada pola DDD. Desain tersebut memiliki lapisan-lapisan seperti yang digambarkan pada Gambar 3.4, dan tiap lapisan memiliki alur *dependency* ke inti lapisan. Satu lapisan dari DDD akan memakai lapisan di bawahnya, dengan begitu pengembang dapat mengabstraksikan model bisnis dari inti ke lapisan luar. Menggunakan desain ini alur kebutuhan komponen program dapat diidentifikasikan dengan mudah, membuat proses pengembangan menjadi lebih efisien. Implementasi dari lapisan-lapisan DDD tersebut dijelaskan pada subbab-subbab di bawah.

3.3.1.1 Domain Layer

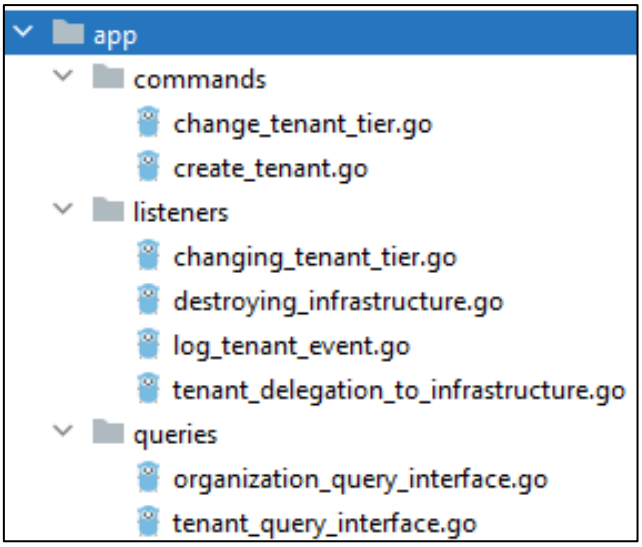
Direktori "domain" merepresentasikan layer "domain" pada DDD. Domain bisnis yang diperlukan adalah sebagai berikut: "entitites" yang bersifat sebagai agregat data; "events" untuk menggambarkan sebuah kejadian dalam proses bisnis di aplikasi yang dijalankan secara asinkron; "repositories" sebagai komponen yang mengurus peredaran data sebuah agregat; "value objects" (dalam konteks ini penulis singkat menjadi "vo") sebagai komponen objek yang merepresentasikan sebuah nilai untuk atribut agregat; dan yang terakhir "services" berisi domain logic yang memiliki ketergantungan dengan servis ekstrenal yang tidak dapat di atributkan kepada "entities" atau "value objects" karena dapat memecahkan isolasi komponen tersebut. Susunan komponen pada lapisan domain diperlihatkan pada Gambar 3.5



Gambar 3.5 Susunan Komponen Domain pada Direktori Kerja Aplikasi

3.3.1.2 Application Layer

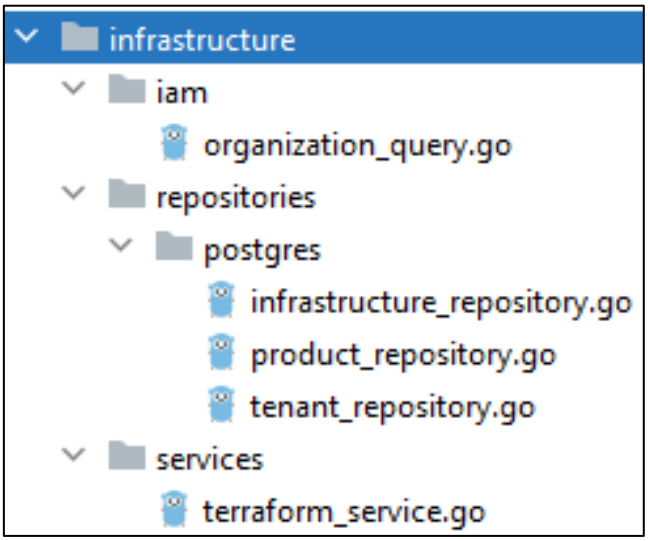
Pemisahan tanggung jawab *Command* dan *Query* dilakukan pada direktori "app" yang merepresentasikan layer "application" pada DDD. Direktori tersebut menampung fitur-fitur yang dapat dipakai pada aplikasi. Direktori "listeners" berisi proses bisnis yang menanggapi "events" yang dipublish baik secara internal dari aplikasi, maupun dari servis eksternal. Direktori "commands" dan "queries" berisi tanggung jawab *read* dan *write* agregat data. Susunan komponen lapisan Application dapat dilihat pada Gambar 3.6.



Gambar 3.6 Isi dari Direktori App Berisi Fitur-Fitur yang Tersedia Pada Aplikasi

3.3.1.3 Infrastructure Layer

Layer "infrastructure" pada DDD direpresentasikan dengan direktori "infrastructures". Di direktori ini akan mengimplementasikan antarmuka yang telah ditulis pada layer "domain" dan "application". Layer ini akan berhubungan langsung dengan servis eksternal di luar aplikasi, biasanya basis data. Dalam konteks tugas akhir ini, servis yang akan dihubungkan adalah basis data menggunakan postgreSQL dan basis data dari wilayah modul IAM. Kumpulan implementasi lapisan Infrastructure dapat dilihat pada Gambar 3.7.



Gambar 3.7 Isi dari Direktori Infrastructure yang Menampung Implementasi Domain

3.3.2 Dependency Injection

Pentingnya menggunakan antarmuka adalah untuk mengurangi *coupling* antara proses bisnis dengan taktik bisnis yang merupakan cerminan dari strategi bisnis, ini berarti kita hanya perlu memikirkan satu hal dalam satu waktu dalam mengembangkan kode kita. Untuk menghubungkan antarmuka dengan implementasi yang ditulis pada "infrastructure" perlu adanya *dependency injection,* hal tersebut dilakukan dalam direktori "dependencies".

Dalam direktori “dependencies” akan dilakukan registrasi kebutuhan untuk dipakai pada komponen lainnya. Registrasi tersebut dilakukan dengan *wrapper function* yang telah dibuat menggunakan pustaka “samber/do” pada Go. Sebagai contoh dalam Kode Sumber 3.1, tipe “INFRA_REPO” dialiaskan sebagai representasi tipe data dari implementasi “InfrastructureRepository" di direktori infrastructure dan Tipe “EVENT_SERVICE” untuk merepresentasikan tipe data “gcp.PubSub”, sebuah package yang penulis buat untuk dapat berkomunikasi dengan event yang dipublish ke PubSub GCP.

```
01    type INFRA_REPO = *postgres.InfrastructureRepository
02    type EVENT_SERVICE = *gcp.PubSub
03
04    func RegisterBindings(app *provider.Application) {
05        infra_repo := postgres.NewInfrastructureService(
06           app.DefaultDatabase(),
07        )
08        event_service := gcp.NewPubSub(app, os.Getenv("MODULE_NAME"))
09
10        provider.Bind[INFRA_REPO](app, infra_repo)
11        provider.Bind[EVENT_SERVICE](app, event_service)
12    }
```

Kode Sumber 3.1 Contoh File bindings.go untuk Mendaftarkan Segala Jenis Kebutuhan pada Aplikasi

Bindings yang telah diregister dalam file “bindings.go” dapat diakses di manapun menggunakan fungsi “Make”. Sebagai contoh pada file “events.go” di Kode Sumber 3.2, yang bertugas meregistrasikan event-event yang dibutuhkan selama alur aplikasi, membutuhkan implementasi dari “EVENT_SERVICE” untuk meregisterkan listener-listener dari sebuah event. Untuk konstruktor listenernya itu sendiri yang memiliki parameter yaitu antarmuka “infrastructure_repository” yang membutuhkan implementasi dari antarmuka tersebut. Dengan menggunakan *wrapper function* yang telah dijahit untuk tugas akhir ini, kebutuhan-kebutuhan tersebut dapat dipenuhi untuk dapat digunakan atau disuntikan kedalam konstruktor listener tersebut.

```
01    func RegisterEvents(app *provider.Application) {
02        infra_repo := provider.Make[INFRA_REPO](app)
03        event_service := provider.Make[EVENT_SERVICE](app)
04        event_service.RegisterListeners(events.INFRASTRUCTURE_DESTROYED,
05           []event.Handler{
06              {
07                 Timeout:  15 * time.Minute,
08                 Listener: listeners.NewDestroyingInfrastructureListener(
09                    infra_repo,
10                 ),
11              },
12           })
13    }
```

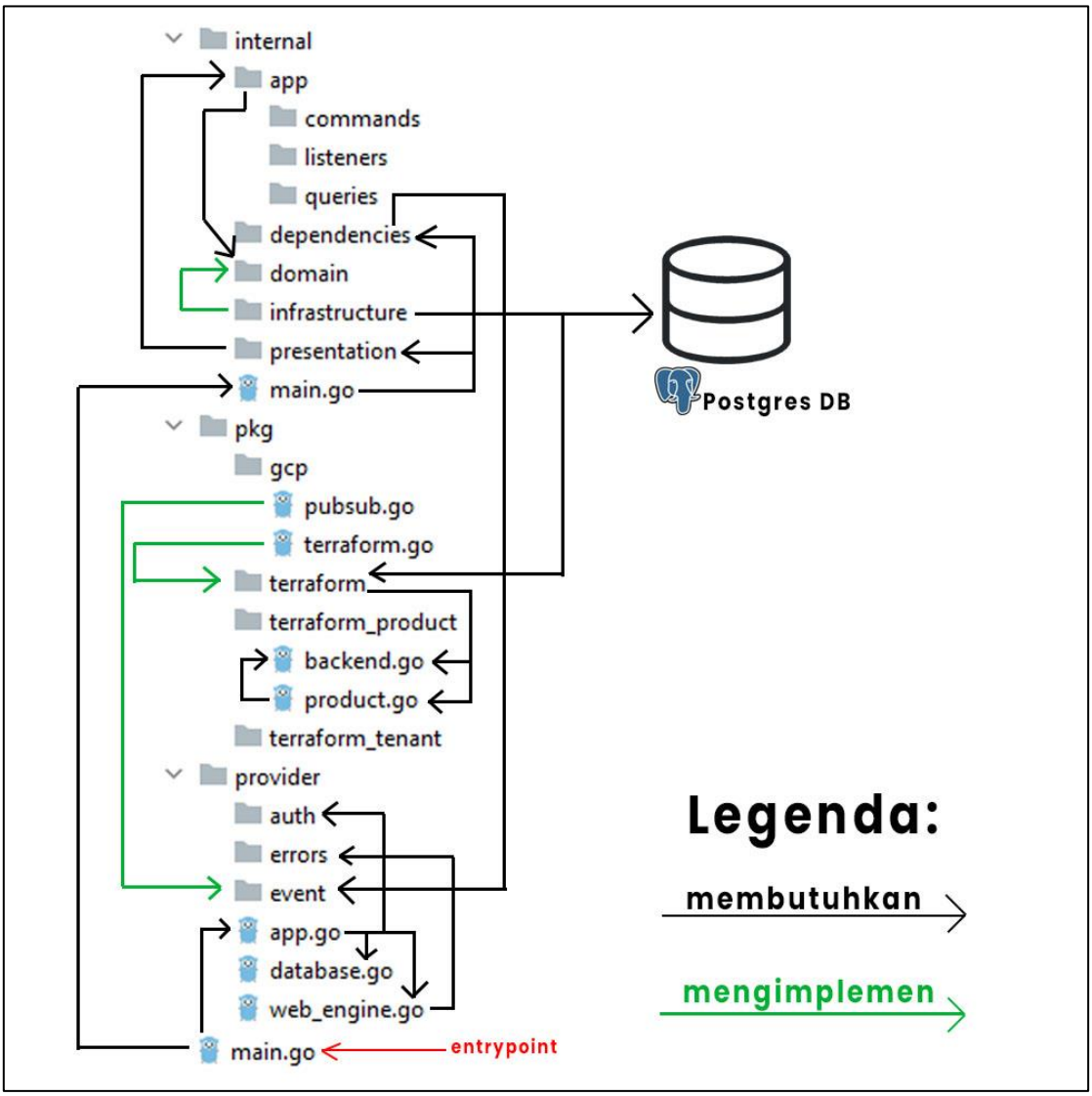
Kode Sumber 3.2 Contoh File events.go untuk Mendaftarkan Jenis Event Apa Saja yang Dipedulikan dan Listenernya

3.3.3 Struktur Aplikasi

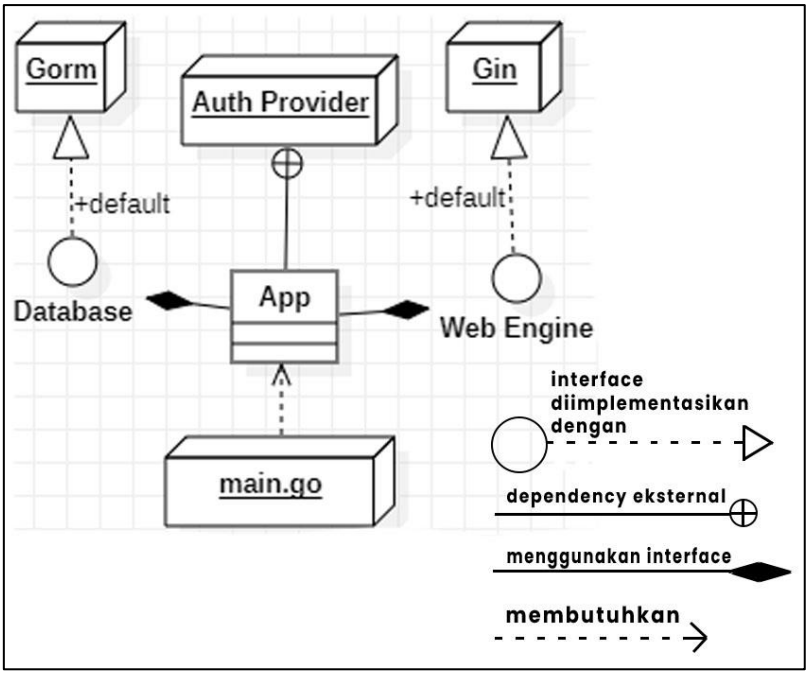
Karena tugas akhir ini bertujuan untuk merancang bangun kerangka kerja, komponen-komponen dalam aplikasi ini didesain agar dapat langsung dipakai atau diganti sesuai kebutuhan, aspek ini dapat dicapai dengan menggunakan antarmuka dan *dependency injection.* Aplikasi ini memiliki beberapa pecahan bagian: “internal” yang berisi segala proses bisnis

aplikasi; “pkg” berisi pustaka-pustaka pendukung yang telah dikembangkan pada tugas akhir ini agar dapat memenuhi konteks aplikasi ini sebagai kerangka kerja manajemen tenant, sekaligus agar pustaka tersebut dapat diimpor oleh developer lain yang ingin menggunakan pustaka ini; “provider” yang bertugas menyediakan platform agar aplikasi ini dapat berjalan.

Untuk menjalankan aplikasi, titik masuk akan dimulai dari “main.go” pada *root* direktori. “main.go” membutuhkan komponen App dari “provider” yang diilustrasikan pada Gambar 3.8. App itu sendiri membutuhkan beberapa komponen yaitu “Database” yang menjadi basis data aplikasi; “Web Engine” yang bertugas dalam menyediakan koneksi web pengguna ke aplikasi; dan “Auth” yang menyediakan layanan autentikasi selama jalannya aplikasi. Sebagai kerangka kerja, semua komponen yang dibutuhkan App dapat diubah oleh pengembang dengan cara mengimplementasikan antarmuka untuk komponen “Database” dan “Web Engine”, dan mendefinisikan url *auth provider* yang akan digunakan pengembang untuk komponen “Auth”. Pengembang juga memiliki pilihan menggunakan implementasi komponen standar yang telah dibuat pada tugas akhir ini (yang diilustrasikan pada Gambar 3.9).

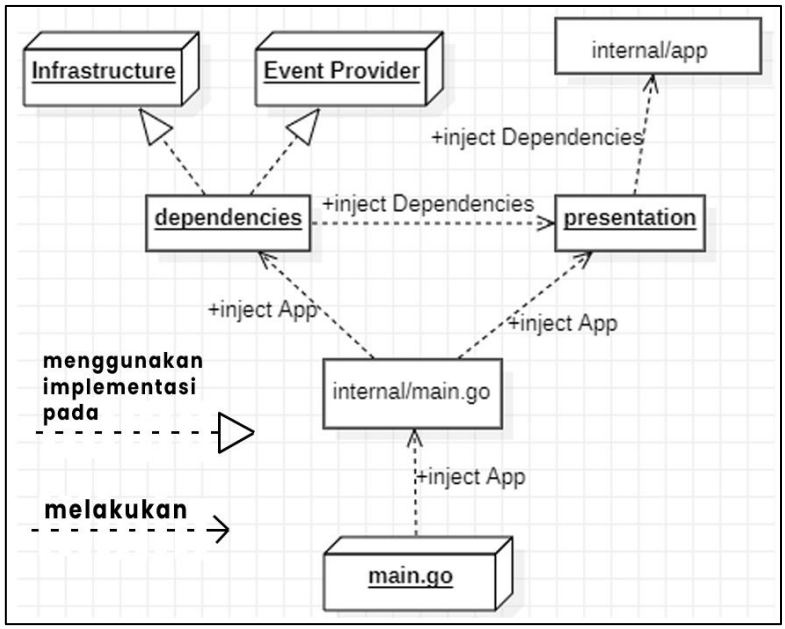


Gambar 3.8 Alur kebutuhan tiap komponen aplikasi



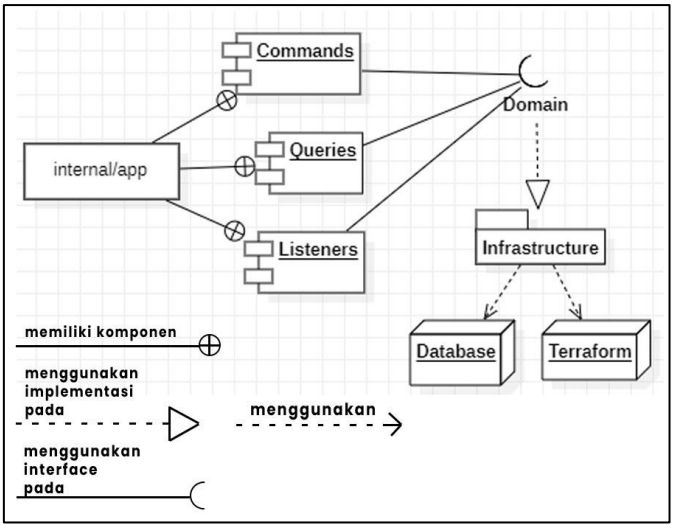
Gambar 3.9 Kebutuhan App untuk Dipakai oleh File main.go

Komponen "main.go" akan menginisialisasi aplikasi dalam direktori "internal" dengan menginjeksikan komponen App yang telah dikonstruksi sebelumnya (Gambar 3.9). Pada titik awal di dalam direktori "internal", proses inisialisasi akan mendaftarkan implementasi antarmuka pada direktori "dependencies" yang menjadi dependensi dan dibutuhkan aplikasi (pada Gambar 3.10). semua dependensi yang telah didaftarkan akan di injeksi ke pada komponen "presentation" yang berisi *controller* yang menerima antarmuka dependensi yang telah di definisi sebelumnya pada "dependencies". *Controller* pada "presentation" akan menggunakan komponen "app" pada direktori "internal" untuk mengontrol proses bisnis tenant, berdasarkan kebutuhan *Command* atau *Query* (yang telah dijelaskan pada subbab 2.12).

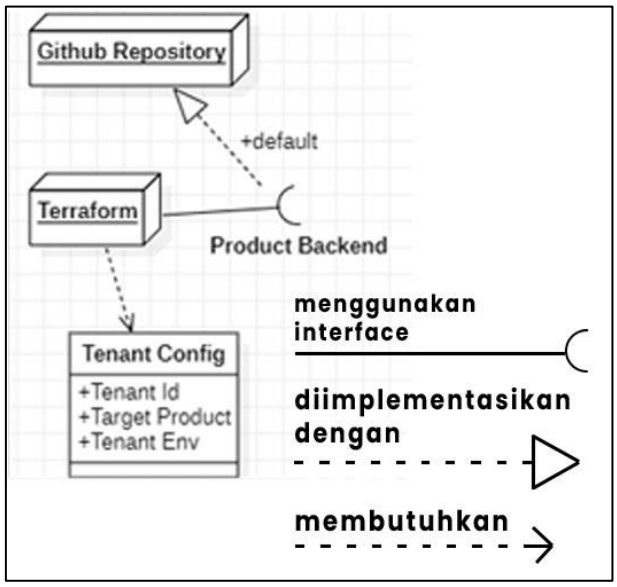


Gambar 3.10 Injeksi App dan Kebutuhan pada Lapisan Presentation

Komponen “app” dalam “internal” menampung proses bisnis aplikasi. Menggunakan arsitektur CQRS, tanggung jawab *read* dan *write* dipisahkan pada direktori “Queries” dan “Commands” dalam “app”. Direktori “listeners” menangani proses bisnis yang menggunakan *event* dan dilakukan secara asinkron. Dalam konteks tugas akhir ini, *event* digunakan untuk melakukan proses aplikasi secara asinkron dan berkomunikasi antar modul tanpa adanya dependensi langsung dengan modul lain untuk situasi yang mengharuskan proses bisnis tenant tidak terhambat oleh proses aplikasi. Komponen “app” menggunakan strategi bisnis yang telah didefinisikan pada komponen “Domain” untuk melayani proses bisnis tenant. Dengan mengutilisasikan *dependency injection*, kebutuhan implementasi antarmuka domain dapat dipenuhi dengan cara menginjeksikan kebutuhan seperti pada Gambar 3.10. Kebutuhan-kebutuhan domain pada Gambar 3.11 yang menggambarkan kebutuhan layer Application pada Domain dan Pkg dan Gambar 3.12 yang menggambarkan Kebutuhan Pkg Terraform kepada Antarmuka Product Backend dan Tenant Config) berupa *instance* basis data dan juga servis eksternal seperti *Terraform*.



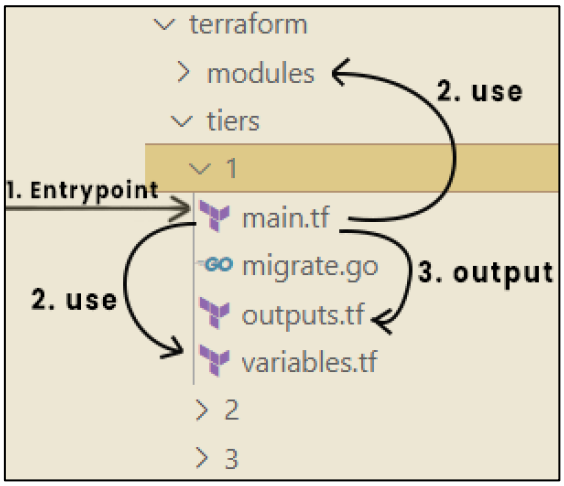
Gambar 3.11 Alur Kebutuhan Layer Application



Gambar 3.12 Alur Kebutuhan Pkg Terraform

3.3.4 Struktur Kode Terraform

Terraform digunakan untuk menerapkan konfigurasi produk yang akan dideploy / *migrate* (memindahkan sumber daya *deployment*). Setiap produk memiliki *repository* config, atau direktori config pada *repository* produk itu sendiri, *repository* tersebut akan dipakai oleh aplikasi SaaS untuk digunakan config produknya. Tiap *repository* config terdapat beberapa direktori yang berisikan *tier* produk. Isi dari tiap direktori *tier* akan secara bebas dikonfigurasi oleh developer yang akan mengembangkan produk SaaS, selama terdapat file "main.tf", "outputs.tf", "variables.tf" dan script "migrate.*". Dalam contoh konfigurasi produk pada Gambar 3.13, penulis memisahkan konfigurasi sumber daya produk yang ingin dipakai menjadi modul, modul-modul tersebut akan dipakai di "main.tf" pada sebuah *tier*.



Gambar 3.13 Alur Penggunaan File Config Terraform dalam Tier

Komponen file dalam direktori Terraform terdiri dari 4 hal utama, "main.tf" yang merupakan entrypoint jalannya script; "migrate.*" merupakan script file untuk melakukan migrasi setiap ada perubahan *tier*; "variables.tf" untuk digunakan selama runtime; dan "outputs.tf" untuk menampung metadata hasil deployment. Variable dapat diisi sebuah value pada saat menjalankan Terraform, dalam konteks tugas akhir ini variable yang selalu diperlukan adalah id infrastruktur (bertipe UUID) dan id provider yang menunjuk id project GCP. Format output akan distandarisasi menjadi dua atribut yang akan dicontohkan pada Kode Sumber 3.3.

```
01    output "resource_information" {
02      value = {
03        serving_url = module.compute.compute_url
04      }
05    }
06    output "metadata" {
07      sensitive = true
08      value = {
09        serving_url = module.compute.compute_url
10        compute = {
11          compute_url = module.compute.compute_url
12        }
13        storage = {
14          host     = module.storage.storage_host
15          port     = module.storage.storage_port
16          name     = module.storage.storage_name
17          user     = module.storage.storage_user
18          password = module.storage.storage_password
19        }
20      }
21    }
```

Kode Sumber 3.3 Contoh Isi File outputs.tf

Hasil atribut “resource_information” pada contoh Kode Sumber 3.3 akan disimpan pada aplikasi SaaS kepada tabel tenant, sebagai entrypoint tenant saat ingin memasuki produk SaaS. Atribut “metadata” akan disimpan pada aplikasi SaaS di metadata pada tabel infrastruktur, yang menjelaskan metadata hasil deployment. Metadata dipakai untuk produk melakukan migrasi dari metadata lama ke metadata baru, aplikasi SaaS secara sederhana akan menyematkan metadata tersebut ke script *migrate* pada *repository* config produk sebagai arguments, dengan metode ini produk dapat mengurus dirinya sendiri tanpa adanya *coupling* dengan aplikasi SaaS.

Script “migrate.*” adalah script migration yang dijalankan setiap adanya perubahan *tier* tenant yang membutuhkan perpindahan data dari sumber daya lama ke sumber daya baru. Script migration tidak terikat oleh bahasa pemrograman spesifik dan usecase yang berarti developer produk dapat membuat script tersebut menggunakan bahasa apapun dan tujuan apapun dalam konteks migrasi tenant. Satu *constraint* pada script migrate tersebut adalah harus menerima argumen dengan key “old” dan “new” dengan tipe data string yang berisikan metadata infrastruktur lama yang dipakai tenant dan metadata infrastruktur baru yang akan dipakai tenant. Dalam konteks tugas akhir ini *script migrate* pada Kode Sumber 3.4 dibuat menggunakan bahasa Go dengan mengikuti kebutuhan yang telah ditetapkan sebelumnya.

```
01    type Infrastructure struct {
02        Host string `json:"host"`
03        Port string `json:"port"`
04        Name string `json:"name"`
05        User string `json:"user"`
06        Password string `json:"password"`
07    }
08
09    func main()error {
10        old_infra := flag.String("old", "", "infrastuktur lama tenant")
11        new_infra := flag.String("new", "", "infrastuktur baru tenant")
12
13        flag.Parse()
14
15        if *old_infra == "" || *new_infra == "" {
16           fmt.Println("argument yang berisi infrastruktur tidak boleh "
+
17              "kosong agar dapat melakukan migrasi data")
18           return
19        }
20
21        var oldInfra, newInfra Infrastructure
22        json.Unmarshal([]byte(*old_infra), &oldInfra)
23        json.Unmarshal([]byte(*new_infra), &newInfra)
24
25        // lakukan migrasi menggunakan
26        // data koneksi DB infrastruktur lama ke baru...
          .
          .
          .
39        return nil
40    }
```

Kode Sumber 3.4 Contoh Script Migration Data Tenant

File “main.tf” menjelaskan konfigurasi apa saja yang dibutuhkan produk untuk dideploy pada provider, dalam konteks tugas akhir ini provider yang akan dipakai adalah GCP. Di *file* ini developer bebas mengonfigurasi kebutuhan produk dengan mengikuti kebutuhan *variables* yang telah ditetapkan dalam konteks tugas akhir ini. Contoh dari file main.tf dan variables.tf diperlihatkan pada Kode Sumber 3.5 dan Kode Sumber 3.6

```
01    provider "google" {
02      project = var.provider_id
03    }
04
05    module "storage" {
06      source = "../../modules/storage"
07
08      storage_instance_name = var.infrastructure_id
09    }
10
11    module "compute" {
12      source = "../../modules/compute"
13
14      # Pool configuration
15      compute_name      = var.infrastructure_id
16      storage_host      = module.storage.storage_host
17      storage_port      = module.storage.storage_port
18      storage_name      = module.storage.storage_name
19      storage_user      = module.storage.storage_user
20      storage_password  = module.storage.storage_password
21    }
```

Kode Sumber 3.5 Contoh Isi File main.tf

```
1    variable "provider_id" {
2      description = "provider project id"
3    }
4
5    variable "infrastructure_id" {
6      description = "infrastructure id"
7    }
```

Kode Sumber 3.6 Contoh Variabel yang Dipakai pada main.tf

3.3.5 Struktur Basis Data

Tabel 3.3 Detail Tabel Tenants

Nama Kolom	Deskripsi	Tipe	Keterangan
id	Identifier sebuah tenant	UUID Versi 4	-
product_id	Foreign Key yang menunjuk ke tabel Products	UUID Versi 4	-
organization_id	Foreign Key yang menunjuk ke tabel Organizations	UUID Versi 4	-

Nama Kolom	Deskripsi	Tipe	Keterangan
infrastructure_id	Foreign key yang menunjuk ke tabel Infrastructures	UUID Versi 4	-
name	Nama dari tenant	String	-
status	Status lifecycle tenant	String	Enum (tenant_onboarding, tenant_migrating, activated, deactivated)
resource_information	Informasi tambahan tenant terhadap produk	Json	Berbeda dari metadata dari tabel infrastructures, informasi ini berhubungan dengan tenant bukan deployment (contoh: tenant icon, tenant app url)
created_at	Menunjuk waktu ketika terbuatnya tenant pada sistem	Timestamp	-
updated_at	Menunjuk waktu ketika data tenant diubah	Timestamp	-
deleted_at	Menunjuk waktu ketika tenant di hapus	Timestamp nullable	Kolom ini digunakan karena sebagai metode soft delete

Tabel 3.4 Detail Tabel Infrastructures

Nama Kolom	Deskripsi	Tipe	Keterangan
id	Identifier sebuah infrastructure yang akan dipakai tenant	UUID Versi 4	-
product_id	Foregin key yang menunjuk ke tabel produk	UUID Versi 4	-
metadata	Informasi mengenai deployment sumber daya	Json	Berisi informasi kredensial sumber daya seperti koneksi database, compute url
user_limit	Limit tenant yang dapat memakai infrastruktur ini	Int	Dikarenakan 1 infrastruktur dapat digunakan oleh banyak tenant (skema *POOL*)
deployment_model	Jenis skema deployment yang digunakan tenant	String	Enum (*POOL*, *SILO*)

Nama Kolom	Deskripsi	Tipe	Keterangan
Prefix	Prefix lokasi penyimpanan terraform state pada provider	String	-
created_at	Menunjuk waktu ketika terbuatnya infrastructure pada sistem	Timestamp	-
updated_at	Menunjuk waktu ketika data infrastructure diubah	Timestamp	-
deleted_at	Menunjuk waktu ketika infrastructure di hapus	Timestamp nullable	Kolom ini digunakan karena sebagai metode soft delete

Tabel 3.5 Detail Tabel Products

Nama Kolom	Deskripsi	Tipe	Keterangan
id	Identifier sebuah products yang akan dipakai tenant	UUID Versi 4	-
app_id	Foreign key yang menunjuk ke tabel Apps	Int	-
deployment_schema	Informasi terkait penyimpanan konfigurasi produk	Json	Isinya berupa informasi lokasi / path / cara untuk mendapatkan konfigurasi produk dari sistem
*tier*_name	Nama *tier* produk	String	Produk menunjuk ke aplikasi
deployment_model	Jenis skema deployment *tier*	String	Enum (*POOL, SILO*)
Price	Harga produl	Int	-
created_at	Menunjuk waktu ketika terbuatnya infrastructure pada sistem	Timestamp	-
updated_at	Menunjuk waktu ketika data infrastructure diubah	Timestamp	-

Tabel 3.6 Detail Tabel Apps

Nama Kolom	Deskripsi	Tipe	Keterangan
id	Identifier sebuah aplikasi yang akan dijual	int	Aplikasi merupakan base produk yang mempunyai banyak *tier* pada tabel Products
Name	Nama aplikasi yang dijual	String	-
Icon	Gambar icon aplikasi	String	-
created_at	Menunjuk waktu ketika terbuatnya infrastructure pada sistem	Timestamp	-
updated_at	Menunjuk waktu ketika data infrastructure diubah	Timestamp	-

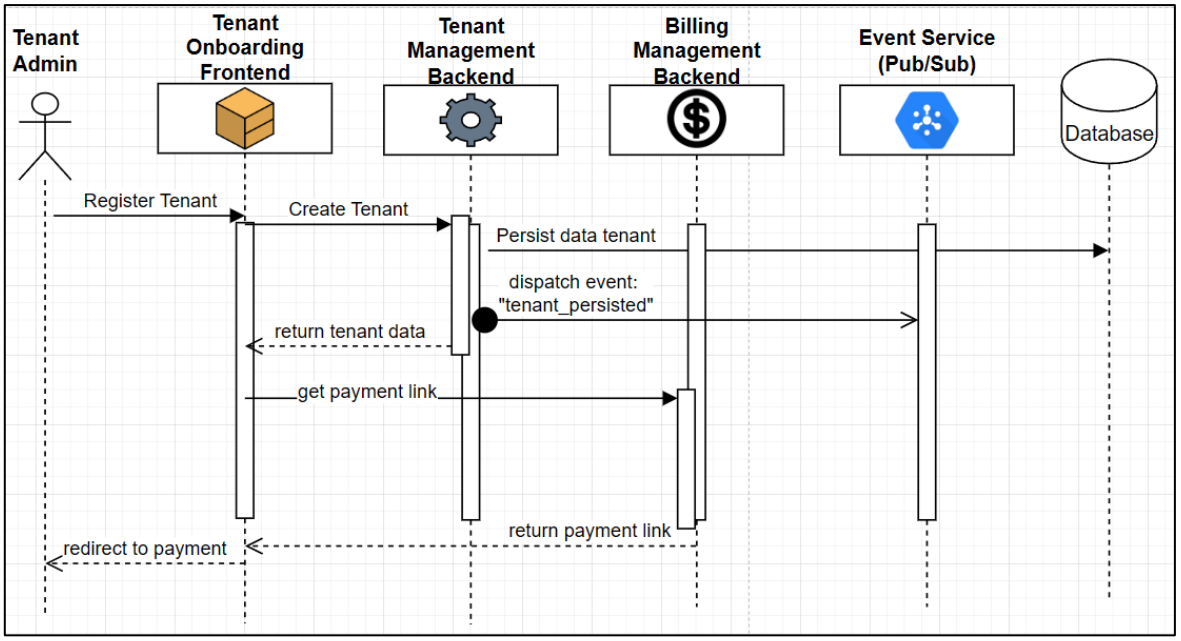
Tabel 3.7 Detail Tabel Organizations

Nama Kolom	Deskripsi	Tipe
id	Identifier origanisasi yang mempunyai kumpulan tenant	UUID Versi 4
name	Nama organisasi	String
created_at	Menunjuk waktu ketika terbuatnya infrastructure pada sistem	Timestamp
updated_at	Menunjuk waktu ketika data infrastructure diubah	Timestamp

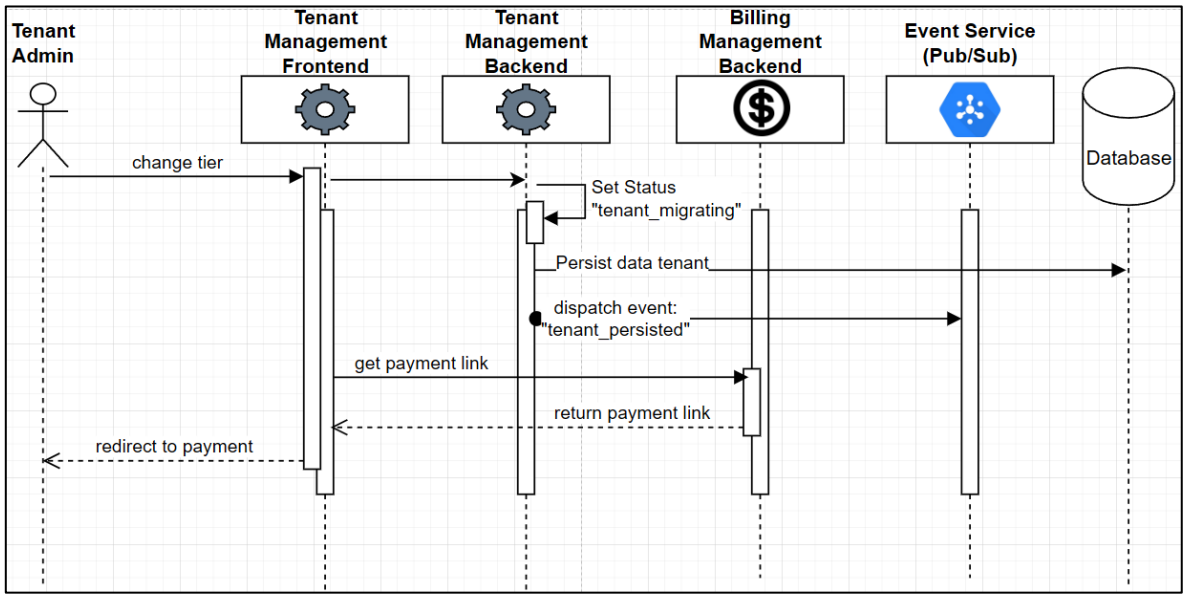
3.3.6 Alur Proses Bisnis *Lifecycle* Tenant

Sebagai modul Tenant Onboarding sistem harus dapat mengelola *lifecycle* atau bisa dibilang sebagai alur hidup tenant. Alur hidup ketika tenant pertama kali mendaftar atau mengubah datanya atau sumber dayanya harus dikelola oleh modul Tenant Management. Sebagai kerangka kerja sistem harus bisa berintegrasi dengan modul atau komponen lainnya dalam aplikasi SaaS. Agar sistem dapat berintegrasi dengan mudah tanpa adanya ketergantungan langsung antar modul yang menyebabkan bertambahnya waktu pengembangan (dan sebagai kerangka kerja bertujuan untuk menurunkan waktu tersebut), maka sistem harus mempunyai ketergantungan yang longgar antar komponen, pada tugas akhir ini aspek tersebut akan diimplementasikan menggunakan sistem *Event Driven Architecture.* Pada alur di Gambar 3.14, Gambar 3.15 diperlihatkan tidak ada nya ketergantungan langsung antara *Backend* dengan *Backend* (karena di situlah proses bisnis dijalankan), semua diperantarai oleh *event service*. Dengan menggunakan metode ini ketergantungan antar modul akan longgar dan modul yang

dikembangkan bisa dipakai semudah *plug and play* dengan modul lain yang pengembang ingin pakai sesuai kebutuhan pengembang SaaS.

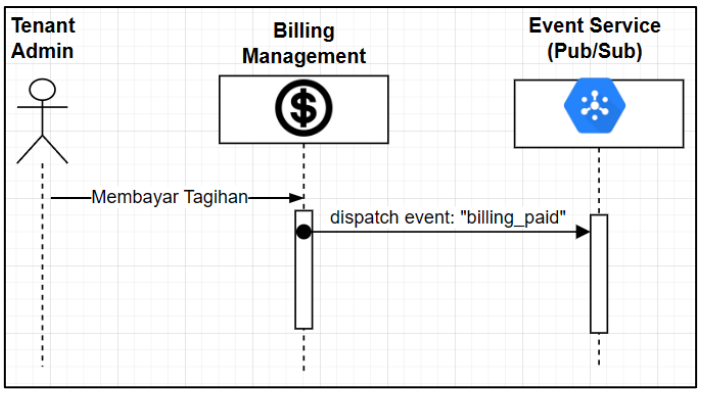


Gambar 3.14 Gambaran Alur Tenant Onboarding saat Membuat Tenant Baru



Gambar 3.15 Gambaran Alur Tenant Mengganti Tier Produk

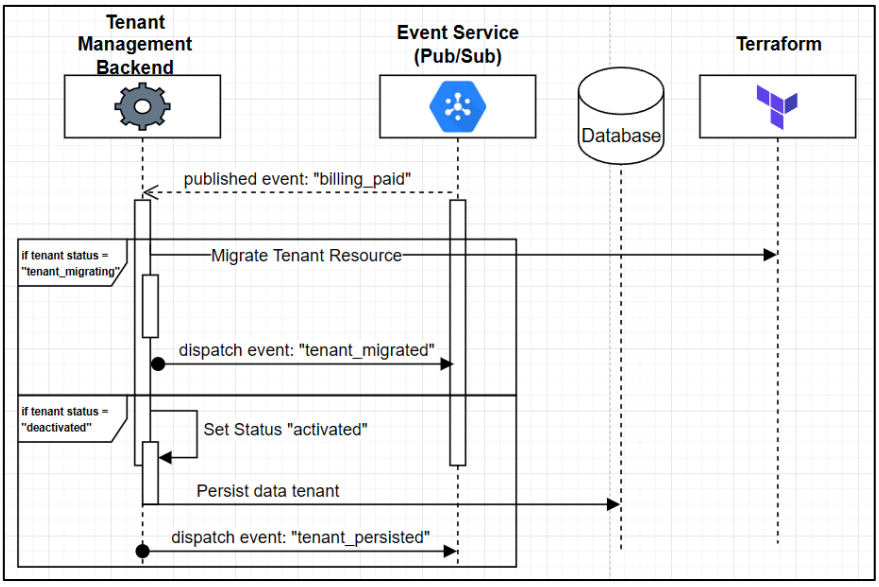
Gambar 3.16 memperlihatkan kasus ketika Billing dibayar oleh Tenant Admin. Modul Billing akan memroses pembayaran secara asinkron dan menerbitkan *event* "billing_paid" yang menjadi pemicu sebuah aksi di modul lain. Pada konteks modul Tenant Management, *event* "billing_paid" akan memicu terjadinya proses migrasi bila status Tenant adalah "migrating" dan memicu terjadinya aktivasi Tenant bila statusnya "deactivated"



Gambar 3.16 Gambaran Alur Tenant Membayar Tagihan

3.3.6.1 Pergantian *Tier*

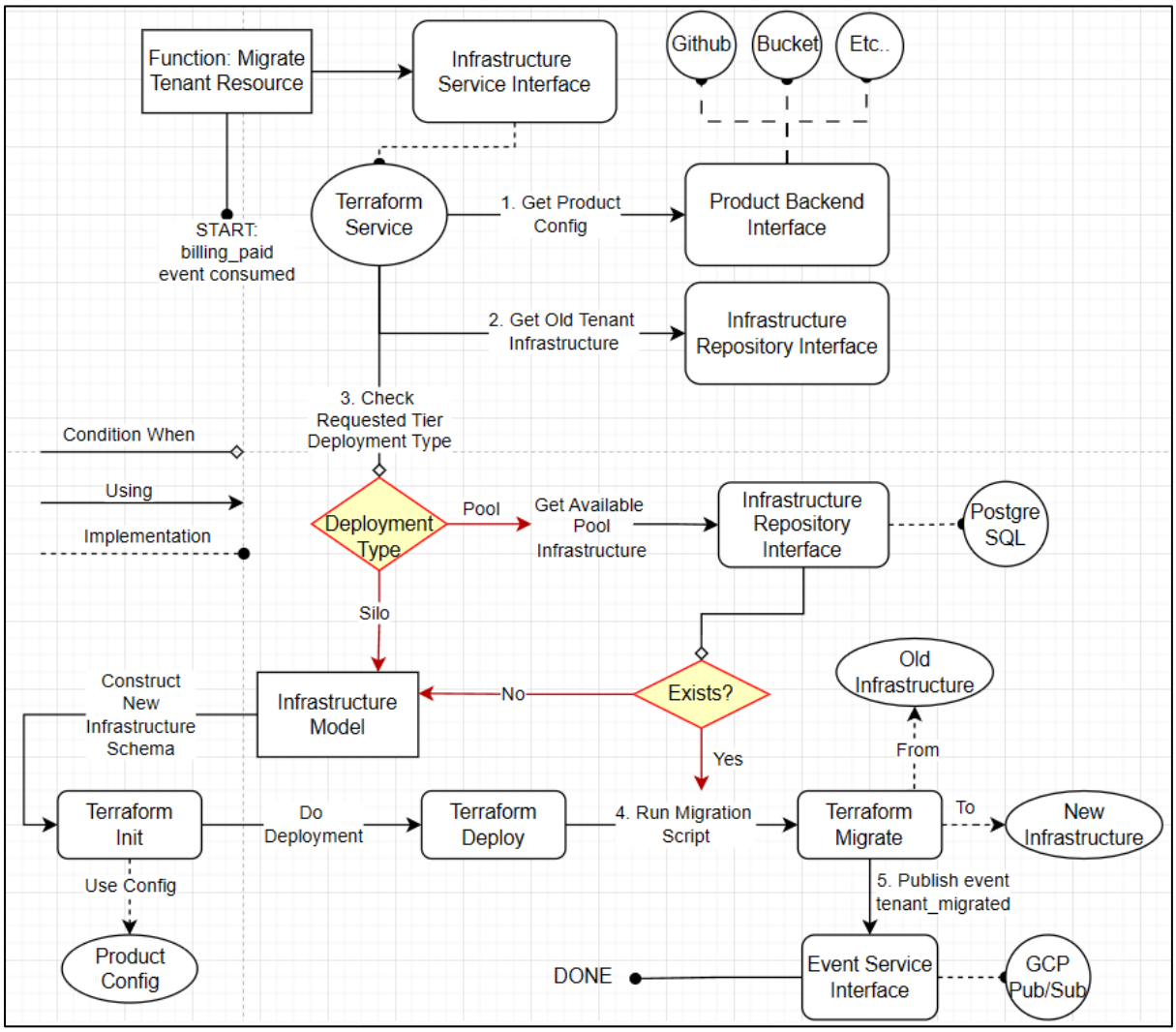
Migrasi sumber daya tenant dilakukan setelah pengguna membayar tagihan dari modul Billing, setelah itu modul billing akan menerbitkan *event* "billing_paid" yang dikonsumsi oleh modul Tenant Management. Setelah pembayaran dikonfirmasi melalui *event* "billing_paid" oleh billing, proses migrasi sumber daya tenant dapat dilakukan jika status tenantnya "tenant_migrating" yang berarti tenant tersebut berada dalam masa migrasi, dengan begitu proses bisnis yang dilakukan adalah konteks memindahkan sumber daya tenant. Status lain yang diterima saat mengonsumsi event "billing" paid adalah jika status tenantnya "deactivated", dalam konteks ini berarti tenant admin melakukan pembayaran dengan tujuan mengaktifkan tenant, maka dari itu proses bisnis yang dilakukan adalah mengaktifkan tenant. Alur dari proses bisnis tersebut dapat dilihat pada Gambar 3.17.



Gambar 3.17 Alur Proses Bisnis Event billing_paid

Pada Gambar 3.18, dijelaskan detail dari mekanisme proses migrasi sumber daya tenant. Alur fungsional migrasi sumber daya tenant dipicu dari terbitnya *event* "billing_paid" oleh modul Billing. Karena tugas akhir ini bertujuan merancang dan membangun kerangka kerja, segala kebutuhan proses bisnis akan dibuat sebagai abstraksi menggunakan antarmuka, dengan cara ini pengembang yang akan menggunakan modul ini untuk mengembangkan aplikasi SaaS

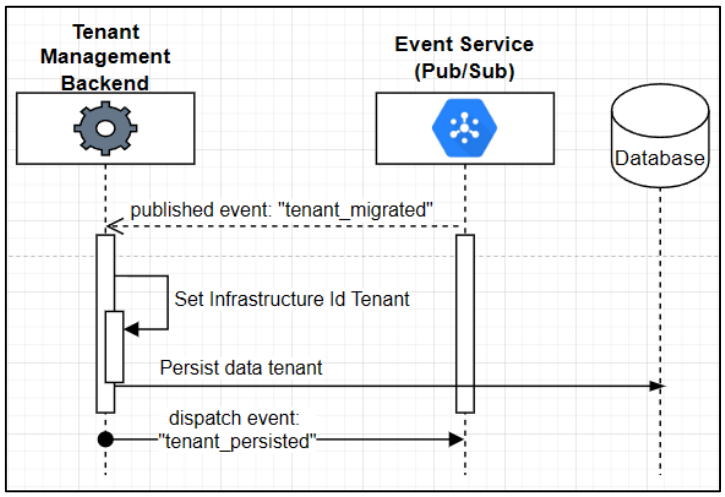
dapat dengan mudah mengimplementasikan kebutuhan proses bisnisnya melalui *dependency injection* dan *domain modeling* yang telah dijelaskan pada landasan teori, dan dijelaskan lebih rinci implementasinya pada subbab implementasi ke depan. Pada tugas akhir ini dalam konteks membangun kerangka kerja, antarmuka Infrastructure Service diimplementasikan menggunakan Terraform. Detail implementasi konkrit migrasi sumber daya di kode sumber dapat dilihat pada Lampiran 2.



Gambar 3.18 Alur Fungsional Migrasi Sumber Daya Tenant

Langkah pertama dari implementasi alur migrasi adalah mengambil data konfigurasi produk yang disimpan secara *remote* di tempat penyimpanan yang diinginkan oleh pengembang, untuk mengambil datanya sudah disiapkan antarmuka yang dapat diimplementasikan pengembang aplikasi SaaS nantinya, dalam konteks tugas akhir ini antarmuka Product Backend akan diimplementasikan menggunakan *repository* Github, karena konfigurasi produk yang dipakai pada tugas akhir ini akan disimpan di *repository* Github. Alur selanjutnya adalah mengambil data infrastruktur yang dipakai tenant saat sebelum melakukan perpindahan *tier*, hal ini dilakukan agar dapat memigrasi data tenant dari infrastruktur lama ke infrastruktur baru yang akan dipakai tenant. Selanjutnya adalah mengecek kondisi tipe deployment *tier* produk yang dipilih tenant, jika tipe nya *POOL* maka sistem akan mencari infrastruktur *POOL* yang masih

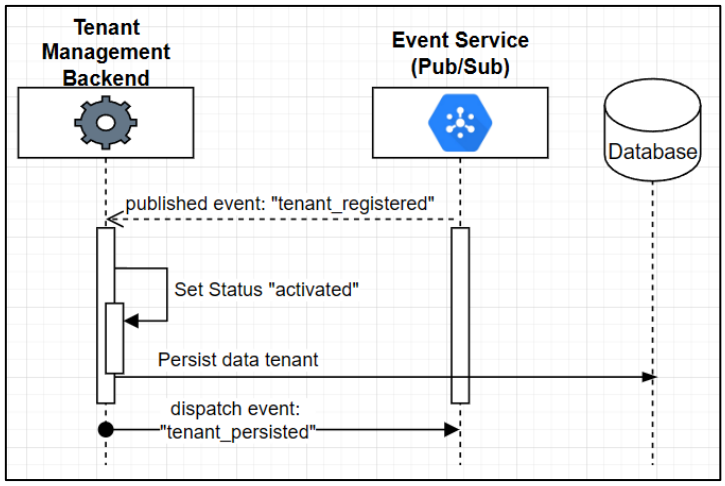
bisa menampung tenant, bila ada maka infrastruktur tersebut akan diterapkan kepada tenant dan langsung dilakukan migrasi data tenant. Bila tipe *tier* produk pilihan tenant adalah *SILO* atau tidak ada infrastruktur *POOL* yang tersedia, maka sistem akan melakukan deployment infrastruktur baru ke provider yang dalam konteks tugas akhir ini akan menggunakan GCP, melalui Terraform Deploy. Setelah infrastruktur sudah diterapkan pada tenant, maka sistem akan melakukan migrasi data tenant, untuk penjelasan lebih rinci mekanisme migrasi akan dijelaskan pada subbab Struktur Kode Terraform. Setelah menjalankan semua proses, maka sistem akan menerbitkan *event* "tenant_migrated" untuk dikonsumsi oleh modul lain atau dalam konteks modul Tenant Management itu sendiri, yang akan melakukan post-migration (diilustrasikan pada Gambar 3.19).



Gambar 3.19 Post-Migration Tenant Memberikan Infrastruktur Sumber Daya Kepada Tenant

3.3.6.2 Aktivasi Tenant

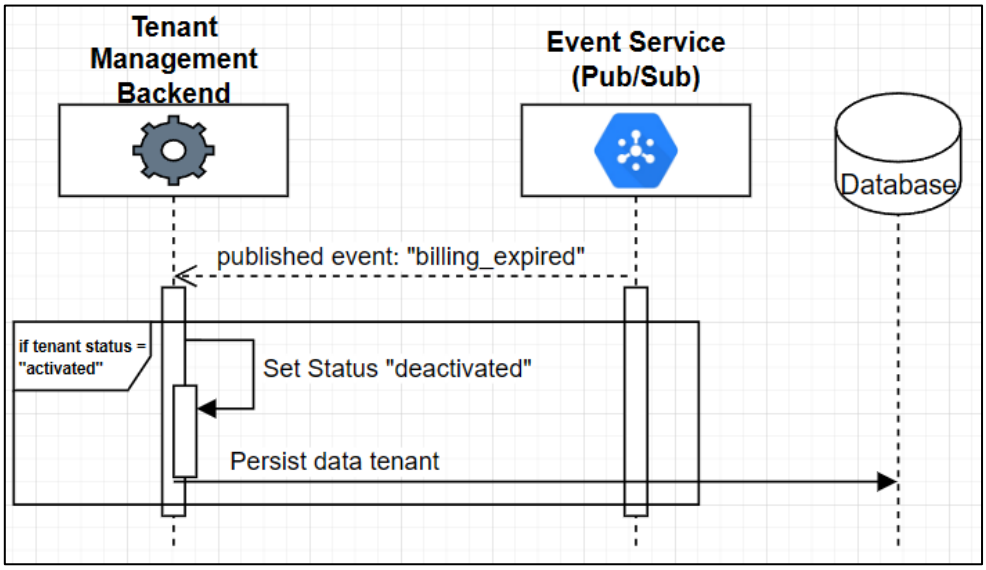
Dalam kasus tenant telah diregistrasikan oleh modul *Identity Management* setelah terjadi perubahan dalam sumber daya setelah proses *onboarding* dan pergantian *tier*, modul *Identity Management* akan menerbitkan *event* "tenant_registrated". Event tersebut dapat diartikan setelah semua proses tenant selesai dari persetujuan proses bisnis modul lain, tenant dapat diaktifkan kembali. Rangkaian kejadiannya diilustrasikan pada Gambar 3.20.



Gambar 3.20 Penanganan Kejadian Tenant Sudah Diregistrasikan

3.3.6.3 Deaktivasi Tenant

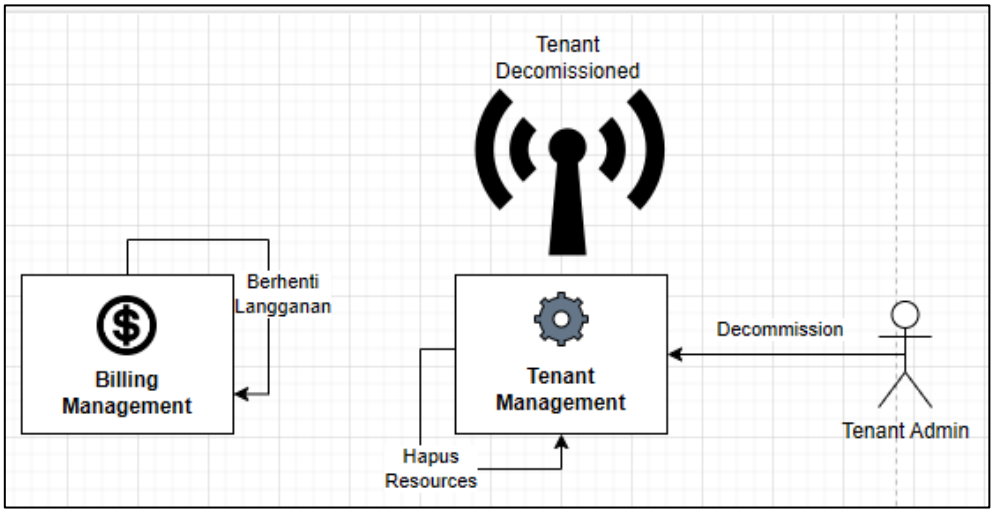
Dalam kasus tenant gagal membayar pada saat statusnya masih “activated” konteks ini berarti sumber daya tenant masih aktif tetapi tenant admin tidak membayar tagihannya tepat waktu, dengan begitu status tenant tersebut akan diubah menjadi “deactivated”. Rangkaian kejadiannya diilustrasikan pada Gambar 3.21.



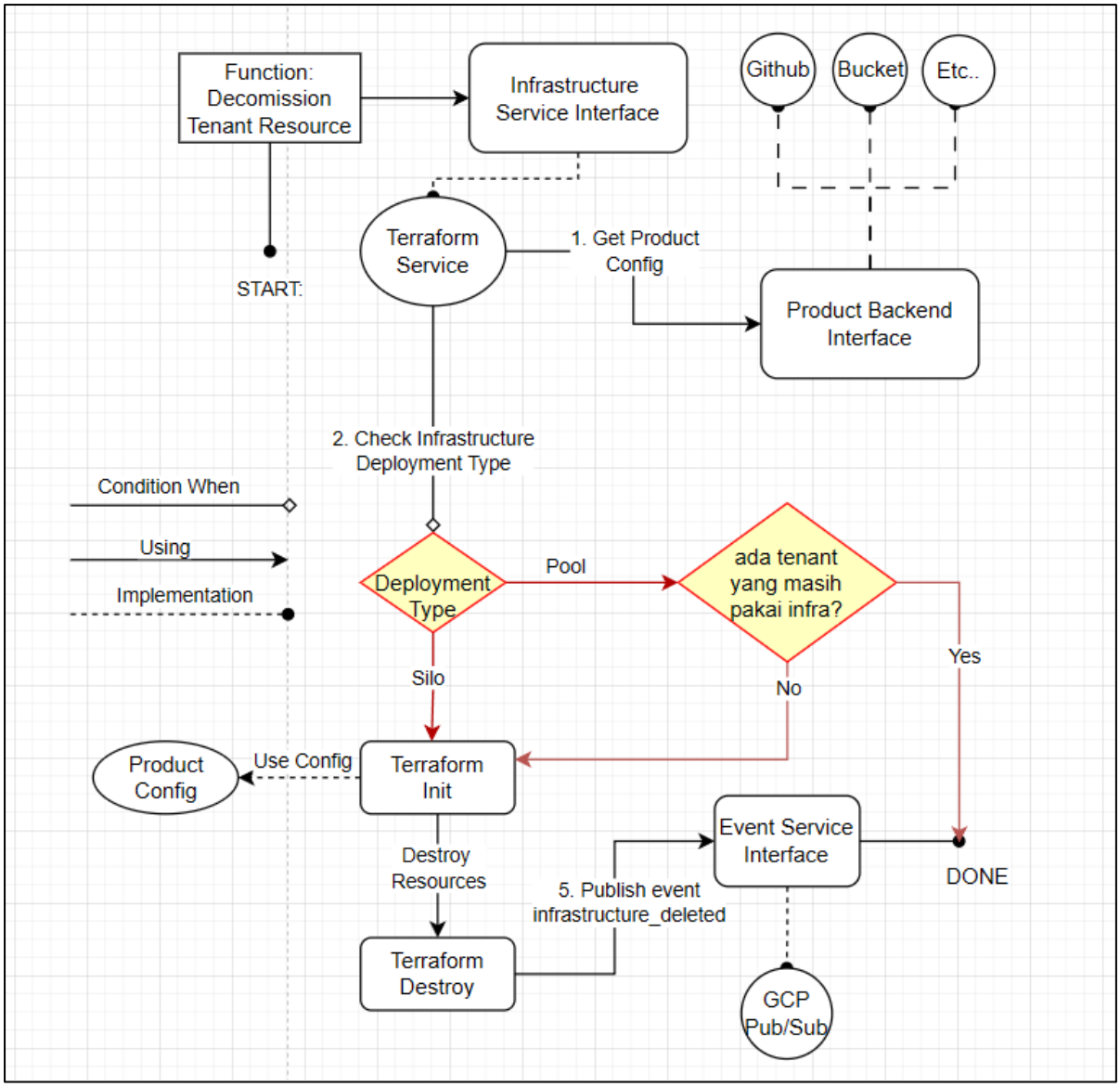
Gambar 3.21 Penanganan Kejadian Tenant Tidak Membayar Billing

3.3.6.4 Dekomisi Tenant

Saat pengguna melakukan dekomisi tenant, berarti pengguna ingin menghilangkan tenant dari organisasi. Hal yang akan dilakukan adalah menghapus data tenant dan menghapus sumber daya pada *cloud provider*, serta menerbitkan event yang akan diproses oleh modul lain yang akan mengurus aksi internal dari masing-masin modul jika tenant terdekomisi. Penggambaran terkait detail proses bisnis dekomisi tenant diilustrasikan pada Gambar 3.22. Detail alur mekanisme dekomisi sumber daya tenant dapat dilihat pada Gambar 3.23. Detail dari implementasi dekomisi sumber daya tenant di kode sumber dapat dilihat pada Lampiran 3.



Gambar 3.22 Ilustrasi Alur Dekomisi Tenant



Gambar 3.23 Detail Mekanisme Alur Dekomisi Sumber Daya Tenant

Mekanisme awal dari proses dekomisi sumber daya tenant hampir mirip dengan proses migrasi sumber daya tenant. Perbedaan proses awal dekomisi dengan migrasi tenant adalah input sistem tidak memerlukan infrastruktur tenant, karena infrastruktur yang akan didekomisi dapat disimpulkan dari infrastruktur tenant yang sekarang. Proses selanjutnya adalah mengecek tipe deployment dari infrastruktur, bila SILO atau POOL dan tidak ada tenant lain yang memakai infrastruktur, langsung lakukan *destroy* menggunakan Terraform, bila POOL tetapi masih ada tenant lain yang memakai infrastruktur tersebut maka infrastruktur sumber daya tidak akan didekomisi. Langkah terakhir adalah menerbitkan event "infrastructure_deleted" ke *event service* sebagai penanda bahwa infrastruktur telah selesai didekomisi. Modul-modul yang memedulikan *event* "infrastructure_deleted" akan memroses secara internal hal yang dibutuhkan setelah terdekomisinya sebuah infrastruktur sumber daya.

3.3.7 Rancangan Endpoint API yang Akan Disediakan

Tabel 3.8 Deskripsi Endpoint API Melihat Daftar Tenant pada Organisasi

API Endpoint	Method	Request Header	Keterangan
/tenant/{organization_id}	GET	Authorization: Bearer {jwt}	Endpoint API untuk mendapatkan daftar tenant pada organisasi

Json Response:	<pre>{
	"code": int,
	"data": [
	{
	"tenant_id": uuid,
	"name": string,
	"status": enum ("activated", "deactivated", "tenant_migrating",
	"tenant_onboarding"),
	"resource_information": null || object<string, string>,
	"product_id": uuid,
	"tier": string,
	"app_id": int,
	"app_name": string
	}
	]
	}</pre>		

Tabel 3.9 Deskripsi Endpoint API Membuat Tenant Baru pada Organisasi

API Endpoint	Method	Request Header	Keterangan
/tenant	POST	Authorization: Bearer {jwt}	Endpoint API untuk membuat tenant baru pada organisasi

Request Body:	<pre>{
	"organization_id": "f755e497-259c-4a5d-8c37-3d4a79f5c3df",
	"product_id": "425a769b-7686-436b-be04-4d9c92176f92",
	"name": "tenant tes"
	}</pre>		

Json Response:	<pre>{
	"code": int,
	"data": {
	"tenant_id": "f6a0504f-31c9-48bf-9cc3-310cc95ea7af",
	"product_id": "425a769b-7686-436b-be04-4d9c92176f92",
	"organization_id": "1d324da0-c36b-4fb2-bea1-19dee034e683",
	"tenant_status": "created",
	"name": "tenant tes"
	},
	"message": "success"
	}</pre>		

Tabel 3.10 Deskripsi Endpoint API Mengubah *Tier* Tenant

API Endpoint	**Method**	**Request Header**	**Keterangan**
/tenant/change_tier	POST	Authorization: Bearer {jwt}	Endpoint API untuk mengganti *tier* tenant

Request Body:	<pre>{
	"tenant_id": "fcc2d9b0-eadc-4894-8674-4a79ec01a66e",
	"new_product_id": "de6e0104-ff2d-4874-ab9a-38852aae50fc"
	}</pre>		

Json Response:	<pre>{
	"code": int,
	"data": {
	"use_billing": bool
	},
	"message": "success"
	}</pre>		

Tabel 3.11 Deskripsi Endpoint API Dekomisi Tenant

API Endpoint	**Method**	**Request Header**	**Keterangan**
/tenant/decommission	PATCH	Authorization: Bearer {jwt}	Endpoint API untuk melakukan dekomisi tenant

Request Body:	<pre>{
	"tenant_id": "1d324da0-c36b-4fb2-bea1-19dee034e683",
	}</pre>		

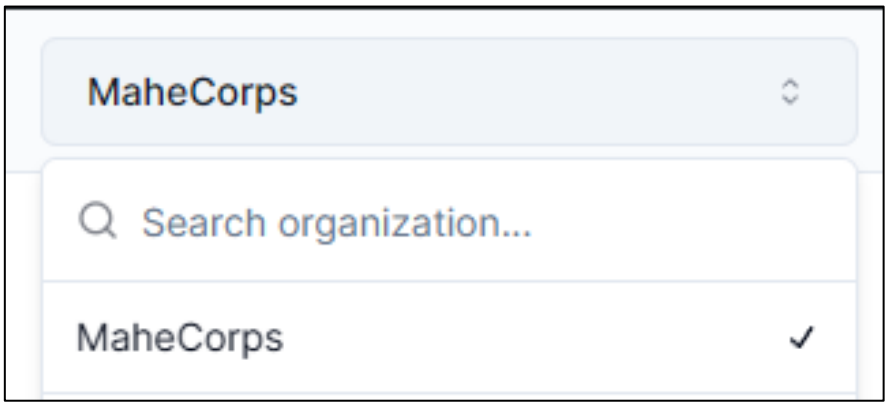
Json Response:	<pre>{
	"code": int,
	"data": null,
	"message": "success"
	}</pre>		

<halaman ini sengaja dikosongkan>

BAB IV HASIL DAN PEMBAHASAN

4.1 Hasil Penelitian

Bagian ini berisikan hasil dari kerangka kerja modul Tenant Management yang telah dirancang dan dibangun untuk aplikasi SaaS dan telah diintegrasikan dengan modul Billing, Onboarding dan Identity Management. Halaman beranda Tenant Management memiliki tampilan daftar tenant yang terasosiasikan dengan organisasi yang dimiliki pengguna aplikasi SaaS. Hasil dari tampilan data organisasi ditunjukan pada Gambar 4.1. Data organisasi yang dimiliki oleh pelanggan SaaS, diambil menggunakan API yang disediakan oleh modul Identity Management. Hasil dari response API list organisasi ditunjukan pada Gambar 4.2, di mana atribut "organization_id" merupakan id dari organisasi Tenant dan "name" merupakan nama Tenant nya.



Gambar 4.1 Hasil Daftar Organisasi yang Dimiliki Pengguna

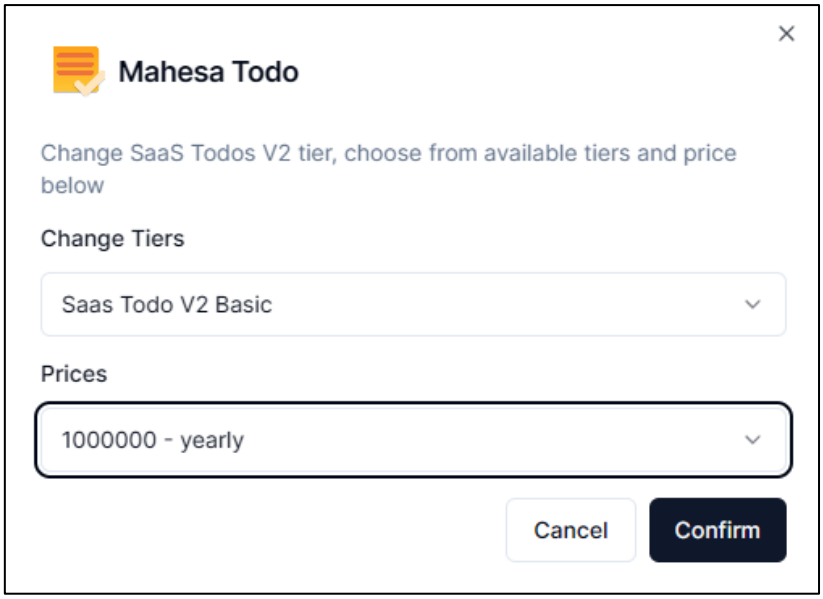

```
{
  "code": 1,
  "data": [
    [
      {
        "organization_id": "71702925-6b85-4446-8589-f0ad0eb9b1da",
        "name": "MaheCorps"
      }
    ]
  ],
  "message": "success"
}
```


Gambar 4.2 Response API Daftar Organisasi yang Dimiliki Pengguna

4.1.1 Modal Pergantian *Tier*

Pengguna aplikasi SaaS atau dalam konteks ini admin tenant dapat mengganti *tier* tenant dengan cara memilih tenant yang ditampilkan pada beranda. Modal pilihan tenant akan muncul di layar dan menampilkan pilihan *tier* tenant yang dapat dipilih, digambarkan pada Gambar 4.3. Response API pada Gambar 4.4 menjelaskan pilihan *tier* dan harga pada modal Gambar 4.3, di mana atribut "data" berisi kumpulan data dari *tier*, yang ditampilkan di modal adalah data "*tier*_name". Masing-masing pilihan *tier* memiliki kumpulan data *prices* yang diambil dari atribut "price" yang berisi kumpulan data price per *tier*. Data yang diambil dari kumpulan data price adalah atribut "price" dan "reccurence" untuk ditampilkan pada modal. Ide dari

masing-masing *tier* dan price akan digunakan sistem untuk melakukan *request* API kepada modul Billing dan Tenant Management.



Gambar 4.3 Modal Pilihan Tier Tenant


```
{
  "data": [
    {
      "id": "8a5f8dc1-b751-4bf1-ad97-42092b50c80a",
      "app_id": "4",
      "name": "SaaS Todos V2",
      "tier_name": "Saas Todo V2 Basic",
      "tier_index": 0,
      "price": [
        {
          "id": "8a5f8dc1-b751-4bf1-ad97-42092b50c80a
          "product_id": "",
          "price": 100000,
          "reccurence": "monthly"
        },
        {
          "id": "8a5f8dc1-b751-4bf1-ad97-42092b50c80a
          "product_id": "",
          "price": 1000000,
          "reccurence": "yearly"
        }
      ]
    },
    {
      "id": "425a769b-7686-436b-be04-4d9c92176f92",
      "app_id": "4",
      "name": "SaaS Todos V2",
      "tier_name": "Saas Todo V2 Premium",
      "tier_index": 1,
      "price": [
        {
          "id": "425a769b-7686-436b-be04-4d9c92176f92",
          "product_id": "",
          "price": 100000,
          "reccurence": "monthly"
        },
        {
          "id": "425a769b-7686-436b-be04-4d9c92176f92",
          "product_id": "",
          "price": 1000000,
          "reccurence": "yearly"
        }
      ]
    }
  ],
  "message": "success",
  "status": 200
}
```


Gambar 4.4 Response API Daftar Harga Produk dari Modul Billing

4.2 Pengujian Fungsional

4.2.1 Pengujian Antarmuka Halaman Autentikasi

Halaman autentikasi aplikasi SaaS modul Tenant Management akan berintegrasi dengan modul *Identity Management* yang akan menyediakan layanan untuk pengguna aplikasi SaaS agar dapat login dan mengakses sumber daya dari organisasinya. Detail dari pengujian antarmuka autentikasi dapat dilihat pada **Tabel 4.1**.

Tabel 4.1 Deskripsi Pengujian P01 Autentikasi

Nomor Pengujian	**P01**
Nama	Pengujian masuk ke beranda dengan belum login
Endpoint	/
Method	GET
Hasil yang diharapkan	Halaman akan didireksikan ke login page provider IAM
Hasil yang didapatkan	Halaman akan didireksikan ke login page provider IAM
Keterangan hasil pengujian	Berhasil

4.2.2 Pengujian Antarmuka Tampilan Halaman Beranda

Halaman beranda menampilkan daftar tenant yang dimiliki organisasi. Daftar tenant didapatkan melalui panggilan API ke sistem *backend* Tenant Management. Detail pengujian antarmuka ditampilkan pada **Tabel 4.*2***.

Tabel 4.2 Deskripsi Pengujian P02 Melihat Daftar Tenant

Nomor Pengujian	**P02**
Nama	Pengujian API melihat daftar tenant dalam organisasi
Endpoint	/api/tenant/1d324da0-c36b-4fb2-bea1-19dee034e683
Method	GET
Hasil yang diharapkan	List data tenant yang terasosiasi dengan id organisasi tersebut
Hasil yang didapatkan	List data tenant yang terasosiasi dengan id organisasi tersebut
Keterangan hasil pengujian	Berhasil

4.2.3 Pengujian Antarmuka Dekomisi Tenant

Tenant dapat diberhentikan melalui dekomisi tenant. Hal ini akan menghapus data tenant dari sistem, serta menghapus sumber daya tenant dari *cloud provider*. Detail dari pengujian antarmuka dekomisi Tenant dipaparkan pada pengujian P03 dalam **Tabel 4. 3**.

Tabel 4. 3 Deskripsi Pengujian Dekomisi Tenant

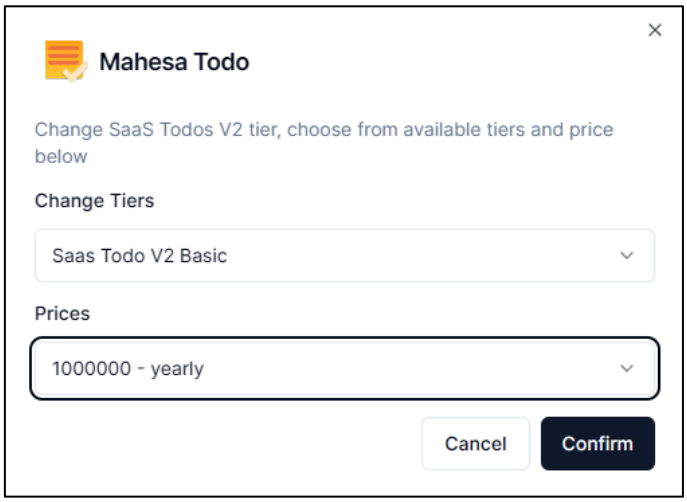
Nomor Pengujian	**P03**
Nama	Pengujian Usecase dekomisi tenant
Endpoint	/tenant/decommission
Method	PATCH
Hasil yang diharapkan	Tenant dan sumber dayanya terhapus dari sistem
Hasil yang didapatkan	Tenant dan sumber dayanya berhasil terhapus dari sistem
Keterangan hasil pengujian	Berhasil

4.2.4 Pengujian Antarmuka Mengganti *Tier* Tenant

Tenant dapat mengganti *tier* tenant dengan memilih pilihan *tier* pada modal yang ditunjukan pada Gambar 4.5. setelah itu tenant akan diarahkan ke laman pembayaran dan status Tenant nya berubah menjadi "migrating". Detail dari pengujian antarmuka mengganti Tenant dipaparkan pada pengujian P04 pada **Tabel 4.4**.

Tabel 4.4 Deskripsi Pengujian P04 Mengganti *Tier* Tenant

Nomor Pengujian	**P04**
Nama	Pengujian Usecase mengganti *tier* tenant dari halaman tenant management
Endpoint	/tenant/change_tier
Method	POST
Hasil yang diharapkan	Status tenant berubah menjadi "migrating", dan mendapat link payment dari Billing
Hasil yang didapatkan	Status tenant berubah menjadi "migrating", tenant diarahkan ke laman pembayaran
Keterangan hasil pengujian	Berhasil



Gambar 4.5 Modal Pilihan Tier Tenant

4.2.5 Pengujian Sistem Penghapusan Sumber Daya Tenant

Sumber daya tenant akan dihapus dari *cloud provider* setelah pengguna melakukan dekomisi Tenant, menggunakan antarmuka yang diuji pada pengujian P04. Detail prosedur dan hasil dari pengujian sistem penghapusan sumber daya Tenant dipaparkan pada **Tabel 4.5**.

Tabel 4.5 Deskripsi Pengujian P05 Menghapus Sumber Daya Tenant

Kode Pengujian	**P05**
Nama	Pengujian penghapusan sumber daya Tenant
Kondisi awal	- Tenant memiliki sumber daya di *cloud provider*
Prosedur pengujian	1. Menerbitkan event "infrastructure_destroyed" untuk infrastruktur 0e38e946-2ac9-4f63-9787-088e71821ef8 2. Modul tenant management menangkap event

	3. Modul tenant management melakukan proses penghapusan sumber daya secara asinkron
Hasil yang diharapkan	- Sumber daya Tenant menghilang dari *cloud provider*
Hasil yang diperoleh	- *Instance cloud run* 0e38e946-2ac9-4f63-9787-088e71821ef8 menghilang dari *cloud provider* - *Instance cloud sql* 0e38e946-2ac9-4f63-9787-088e71821ef8 menghilang dari *cloud provider*
Hasil pengujian	Berhasil

Pada Gambar 4.6 ditunjukan proses penerbitan *event* "infrastructure_destroyed" yang akan memicu terjadinya proses penghapusan sumber daya pada *cloud provider*. Modul Tenant Management berhasil menangkap *event* tersebut dibuktikan dari jalannya proses penghapusan sumber daya yang ditunjukan dari hasil *Logging server*. Jika proses penghapusan sumber daya berhasil, hasil *Logging server* akan mengeluarkan pesan "Destroy Complete" dan menunjukan *output* Terraform yang telah ditetapkan pada subbab 3.3.4.

Gambar 4.6 Proses Penerbitan Event infrastructure_destroyed untuk Memicu Penghapusan Sumber Daya

4.2.6 Pengujian Sistem Migrasi Sumber Daya Tenant dari POOL ke SILO

Setelah tenant membayar tagihan secara asinkron, event "billing_paid" akan diterbitkan dari modul yang menangani pembayaran. Event tersebut akan memicu proses bisnis migrasi sumber daya tenant yang akan dijalankan pada sistem modul Tenant Management secara asinkron sehingga tidak mengganggu proses bisnis tenant lainnya. Pengujian akan dilakukan dengan cara mempublish event dari Pub/Sub untuk mengsimulasi pembayaran tenant. Event tersebut akan ditangkap oleh modul Tenant Management yang akan melakukan migrasi sumber daya.

Tabel 4.6 Deskripsi Pengujian P06 Migrasi Sumber Daya Tenant dari POOL ke SILO

Kode Pengujian	**P06**
Nama	Pengujian migrasi tenant dari sumber daya *POOL* ke *SILO*
Kondisi awal	- Status tenant adalah "migrating" - Sumber daya tenant bertipe *POOL*
Prosedur pengujian	4. Menerbitkan event "billing_paid" 5. Modul tenant management menangkap event 6. Modul tenant management melakukan proses deployment sumber daya 7. Modul tenant management melakukan proses migrasi tenant 8. Modul tenant management menerbitkan event "tenant_migrated" 9. Modul tenant management melakukan proses delegasi infrastruktur baru ke tenant
Hasil yang diharapkan	- Sumber daya tenant berubah menjadi infrastruktur yang baru - Status tenant berubah menjadi "activated"
Hasil yang diperoleh	- Sumber daya tenant berubah dari infrastructure "0e38e946-2ac9-4f63-9787-088e71821ef8" (*POOL*) menjadi infrastructure "f2a04530-1bd3-40e1-a8a3-5202709c5a19" (*SILO*) - Status tenant berubah menjadi "activated"
Hasil pengujian	Berhasil

Detail langkah-langkah yang dilakukan untuk pengujian migrasi sumber daya tenant dari *POOL* ke *SILO* seperti yang telah dipaparkan pada **Tabel 4.6** adalah sebagai berikut: yang pertama melakukan publish event dari event "billing_paid" service Pub/Sub, yang diperlihatkan pada Gambar 4.7; Modul Tenant Management akan menangkap event "billing_paid" yang telah diterbitkan sebelumnya, dan memulai proses deployment ke GCP dan migrasi.

Gambar 4.7 Pengujian Publish Event billing_paid dari Pub/Sub

4.2.7 Pengujian Sistem Migrasi Sumber Daya Tenant Dari SILO ke POOL

Setelah tenant membayar tagihan secara asinkron, event "billing_paid" akan diterbitkan dari modul yang menangani pembayaran. Event tersebut akan memicu proses bisnis migrasi sumber daya tenant yang akan dijalankan pada sistem modul Tenant Management secara asinkron sehingga tidak mengganggu proses bisnis tenant lainnya.

Pengujian akan dilakukan dengan cara mempublish event dari Pub/Sub untuk mengsimulasi pembayaran tenant. Event tersebut akan ditangkap oleh modul Tenant Management yang akan melakukan migrasi sumber daya. Mekanisme migrasi sumber daya dari *SILO* ke *POOL* agak berbeda dengan mekanisme migrasi sumber daya *POOL* ke *SILO*. Pada proses bisnis ini, sistem akan mencari terlebih dahulu infrastruktur *POOL* yang masih memiliki slot untuk bisa dipakai tenant. Bila ditemukan infrastruktur *POOL* dengan slot yang masih kosong ditemukan, sistem tidak akan melakukan proses *deployment* ke GCP, melainkan langsung melakukan proses migrasi data tenant. Bila tidak ditemukan infrasturktur *POOL* dengan slot yang kosong, maka sistem akan melakukan proses deployment terlebih dahulu ke GCP, serupa dengan proses migrasi sumber daya *POOL* ke *SILO.*

Tabel 4.7 Deskripsi Pengujian P07 Migrasi Sumber Daya Tenant dari SILO ke POOL

Kode Pengujian	**P07**
Nama	Pengujian migrasi tenant dari sumber daya *SILO* ke *POOL*
Kondisi awal	- Status tenant adalah "migrating" - Sumber daya tenant bertipe *POOL*
Prosedur pengujian	1. Menerbitkan event "billing_paid" 2. Modul tenant management menangkap event 3. Modul tenant management melakukan pengecekan terhadap infrastruktur POOL yang bisa dipakai tenant 4. Modul tenant management melakukan proses migrasi tenant 5. Modul tenant management menerbitkan event "tenant_migrated" 6. Modul tenant management melakukan proses delegasi infrastruktur baru ke tenant
Hasil yang diharapkan	- Sumber daya tenant berubah menjadi infrastruktur yang baru - Status tenant berubah menjadi "activated"
Hasil yang diperoleh	- Sumber daya tenant berubah dari infrastructure "f2a04530-1bd3-40e1-a8a3-5202709c5a19" (*SILO*) menjadi infrastructure "0e38e946-2ac9-4f63-9787-088e71821ef8" (*POOL*) - Status tenant berubah menjadi "activated"
Hasil pengujian	Berhasil

Detail langkah-langkah yang dilakukan untuk pengujian migrasi sumber daya tenant dari *POOL* ke *SILO* seperti yang telah dipaparkan pada **Tabel 4.7** adalah sebagai berikut: yang pertama melakukan publish event dari event "billing_paid" service Pub/Sub, yang diperlihatkan pada Gambar 4.8; Modul Tenant Management akan menangkap event "billing_paid" yang telah diterbitkan sebelumnnya, dan memulai proses migrasi.

Gambar 4.8 Pengujian Publish Event billing_paid dari Pub/Sub

4.3 Pengujian Refaktorisasi Kerangka Kerja Modul Tenant Management Tanpa Menggunakan Modul Billing dan IAM untuk Melakukan Migrasi Sumber Daya

Refaktorisasi dilakukan dengan cara menambahkan beberapa bagian kecil dalam kode sumber seperti yang ditunjukan pada Gambar 4.9 di mana model Tenant pada sistem Tenant Management akan mempublish event "tenant_migrating_independently" yang akan menjadikan modul Tenant Mangement sebagai pemicu utama dalam melakukan migrasi sumber daya tenant. Metode ini memungkinkan modul Tenant Management melakukan migrasi tanpa adanya modul pemicu yang awalnya adalah Billing yang menerbitkan event "billing_paid" dan modul IAM yang menerbitkan event "tenant_registered".

Refaktorisasi *event listener* pada file "events.go" dilakukan untuk menangkap *event* baru yang dibuat pada Gambar 4.9 agar dapat melakukan migrasi sumber daya secara mandiri. *Event* "tenant_migrating_independently" akan didengarkan oleh modul Tenant Management dan akan melakukan migrasi sumber daya seperti biasa. Refaktorisasi kode sumber dapat dilihat pada Gambar 4.10. Ketergantungan ke modul IAM sudah digantikan dengan *event* "tenant_registered" yang telah diterbitkan dari modul IAM sendiri pada Gambar 4.10.

```
func (t *Tenant) ChangeTier(new_product_id vo.ProductId)
error {
    if t.TenantStatus != TENANT_ACTIVATED {
        return fmt.Errorf("status tenant tidak aktif")
    }
    t.ProductId = new_product_id
    t.TenantStatus = TENANT_MIGRATING

    return nil
}

func (t *Tenant) ActivateWithNewResourceInformation(
resource_info []byte) error {
    if t.TenantStatus != TENANT_MIGRATING {
        return fmt.Errorf("tenant tidak dalam masa migrasi
            resource")
    }
    t.ResourceInformation = resource_info
    t.TenantStatus = TENANT_ACTIVATED
    return nil
}

func (t *Tenant) DelegateNewInfrastructure(new_infra
*Infrastructure.Infrastructure) error {
    if t.TenantStatus != TENANT_MIGRATING {
        return fmt.Errorf("tenant tidak dalam masa migrasi
            resource")
    }
    t.InfrastructureId = new_infra.InfrastructureId
    return nil
}
```

```
func (t *Tenant) ChangeTier(new_product_id vo.ProductId)
error {
    if t.TenantStatus != TENANT_ACTIVATED {
        return fmt.Errorf("status tenant tidak aktif")
    }
    t.ProductId = new_product_id
    t.TenantStatus = TENANT_MIGRATING

    t.Events["tenant_migrating_independently"] = events.
        NewTenantMigratingIndependently(
            t.TenantId.String(),
            t.ProductId.String(),
        )
    return nil
}

func (t *Tenant) DelegateNewInfrastructure(new_infra
*Infrastructure.Infrastructure) error {
    if t.TenantStatus != TENANT_MIGRATING {
        return fmt.Errorf("tenant tidak dalam masa migrasi
            resource")
    }
    t.InfrastructureId = new_infra.InfrastructureId
    t.Events[events.TENANT_MIGRATED] = events.
        NewTenantDelegatedToNewInfrastructure(
            t.TenantId.String(),
            t.InfrastructureId.String(),
            new_infra.Metadata,
        )
    return nil
}
```

Gambar 4.9 Perubahan Kode Sumber pada Model Tenant

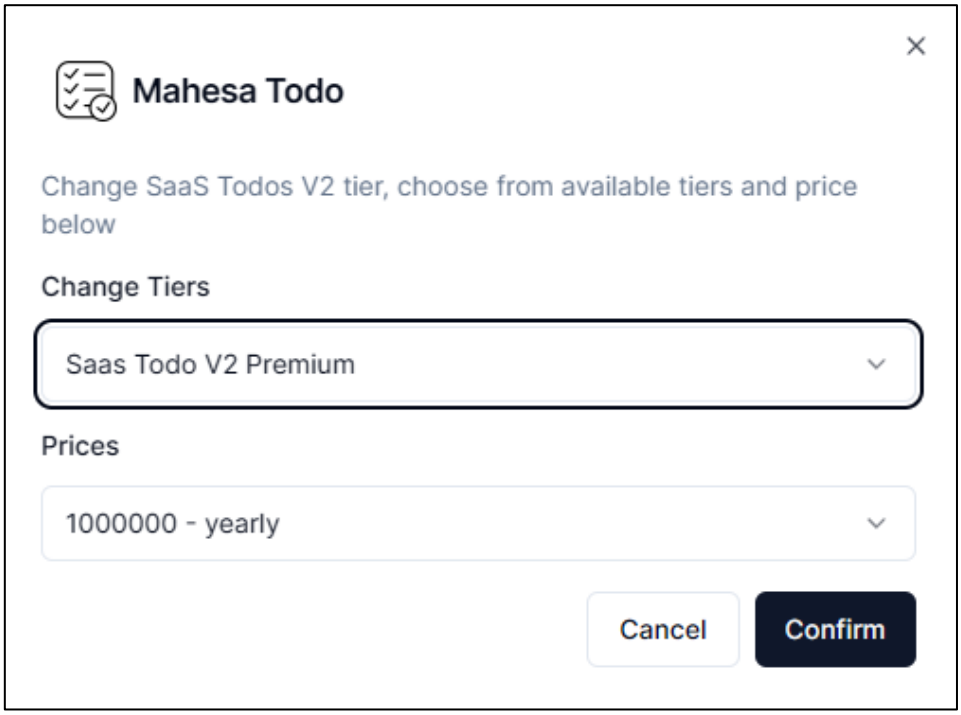
```
func RegisterEvents(app *provider.Application) {
    infra_repo := provider.Make[INFRA_REPO](app)
    tenant_repo := provider.Make[TENANT_REPO](app)
    product_repo := provider.Make[PRODUCT_REPO](app)

    deployer_service := provider.Make[DEPLOYER_SERVICE](app)
    event_service := provider.Make[EVENT_SERVICE](app)

    // akan melakukan migrasi tenant
    event_service.RegisterListeners(events.BILLING_PAID,
        []event.Handler{
            {
                Timeout:  15 * time.Minute,
                Listener: listeners.
                    NewTenantTierChangedListener(product_repo,
                    infra_repo, tenant_repo, deployer_service),
            },
            {
                Listener: listeners.NewActivatingTenang(
                    tenant_repo),
            },
            {
                Listener: listeners.LogTenantEvent(tenant_repo),
            },
        })
    // akan mengubah resource_information tenant dan
    membuat tenant aktif kembali
    event_service.RegisterListeners(events.BILLING_PAID,
        []event.Handler{
            {
                Timeout:  15 * time.Minute,
                Listener: listeners.
                    NewTenantTierChangedListener(product_repo,
                    infra_repo, tenant_repo, deployer_service),
            },
            {
                Listener: listeners.LogTenantEvent(tenant_repo),
            },
        })
    // akan melakukan migrasi tenant tanpa trigger dari
    modul billing
    // dan registrasi tenant resource dari IAM
    event_service.RegisterListeners(
        "tenant_migrating_independently", []event.Handler{
            {
                Timeout:  15 * time.Minute,
                Listener: listeners.
                    NewTenantTierChangedListener(product_repo,
                    infra_repo, tenant_repo, deployer_service),
            },
        })
}
RegisterEvents(app *provider.Application)
```

Gambar 4.10 Meregister Event Baru Yang Akan Melakukan Migrasi Sumber Daya Secara Mandiri

Tabel 4.8 Pengujian P08 Migrasi Sumber Daya Tenant Tanpa Modul Billing

Kode Pengujian	**P08**
Nama	Pengujian migrasi tenant dari sumber daya *SILO* ke *POOL* tanpa menggunakan Billing
Kondisi awal	- Status tenant adalah "migrating" - Sumber daya tenant bertipe *POOL*
Prosedur pengujian	1. Mengubah *tier* tenant dari laman beranda Tenant Management 2. Modul tenant management menangkap event 3. Modul tenant management melakukan proses migrasi tenant 4. Modul tenant management melakukan proses delegasi infrastruktur baru ke tenant
Hasil yang diharapkan	- Sumber daya tenant berubah menjadi infrastruktur yang baru - Status tenant berubah menjadi "activated"
Hasil yang diperoleh	- Sumber daya tenant berubah dari - Status tenant berubah menjadi "activated"
Hasil pengujian	Berhasil

Langkah yang harus dilakukan untuk prosedur pertama pada pengujian P07 adalah memilih *tier* tenant pada laman beranda pada modal pemilihan *tier* yang digambarkan pada Gambar 4.11. Karena dalam konteks ini modul Tenant Management tidak berkolaborasi dengan modul Billing dan IAM, maka tidak perlu melakukan pembayaran dan mengonsumsi *event* dari kedua modul tersebut.

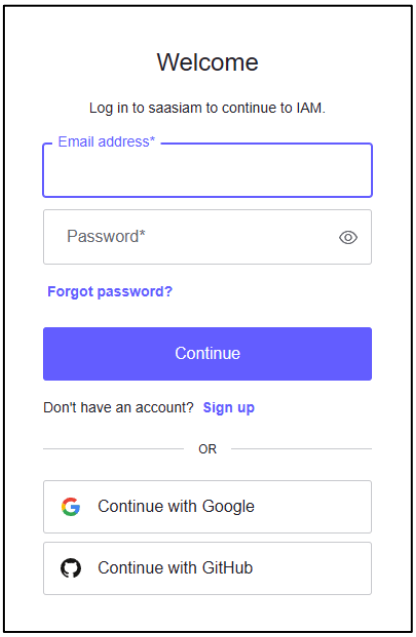


Gambar 4.11 Memilih Tier untuk Secara Langsung Dimigrasi

4.4 Hasil Pengujian

4.4.1 Hasil Pengujian Antarmuka Halaman Autentikasi

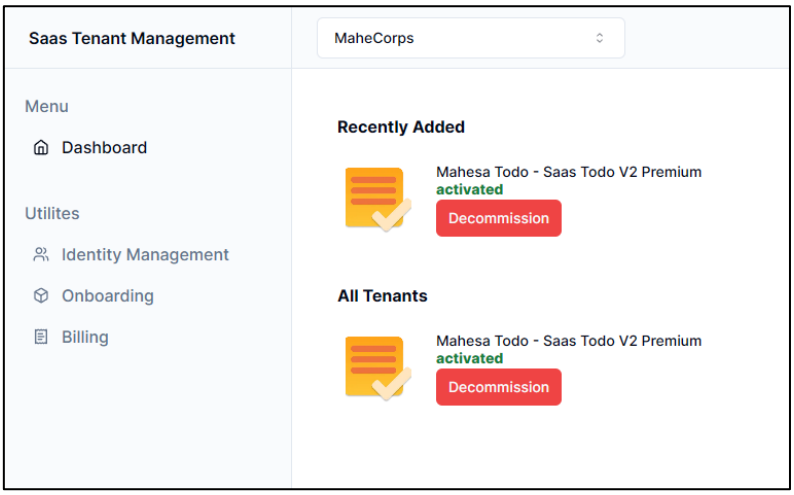
Hasil dari pengujian autentikasi dengan mengakses antarmuka beranda dapat dilihat pada Gambar 4.12. Halaman berhasil diarahkan ke halaman login pada modul IAM.



Gambar 4.12 Hasil Pengujian P01

4.4.2 Hasil Pengujian Antarmuka Tampilan Halaman Beranda

Hasil dari pengujian menampilkan data Tenant dengan mengakses antarmuka halaman beranda dapat dilihat pada Gambar 4.13. Pemanggilan data Tenant melalui *backend* modul Tenant Management berhasil, dan *frontend* dapat menampilkan data yang didapat.

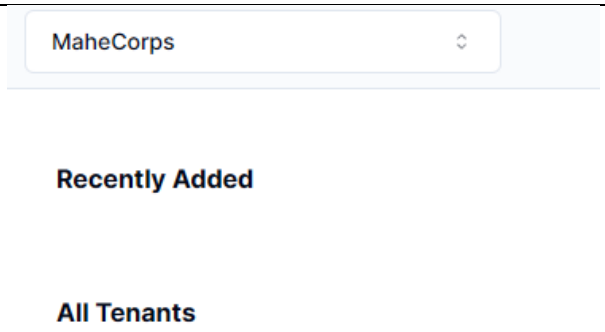


Gambar 4.13 Hasil Tampilan Tenant pada Antarmuka Beranda untuk Pengujian P02

4.4.3 Hasil Pengujian Antarmuka Dekomisi Tenant

Hasil dari pengujian dekomisi Tenant melalui antarmuka ditampilkan pada **Tabel 4. 9**. Hasil pada tabel menampilkan perbandingan perubahan kondisi Tenant pada antarmuka sebelum melakukan dekomisi, dan setelah melakukan dekomisi.

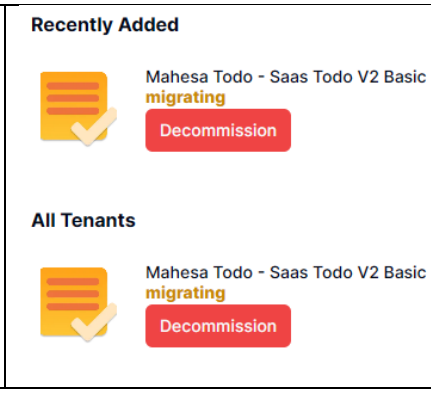
Tabel 4. 9 Hasil Melakukan Dekomisi Tenant

Kondisi Tenant Awal pada Antarmuka Beranda	
Kondisi Tenant Akhir pada Antarmuka Beranda	

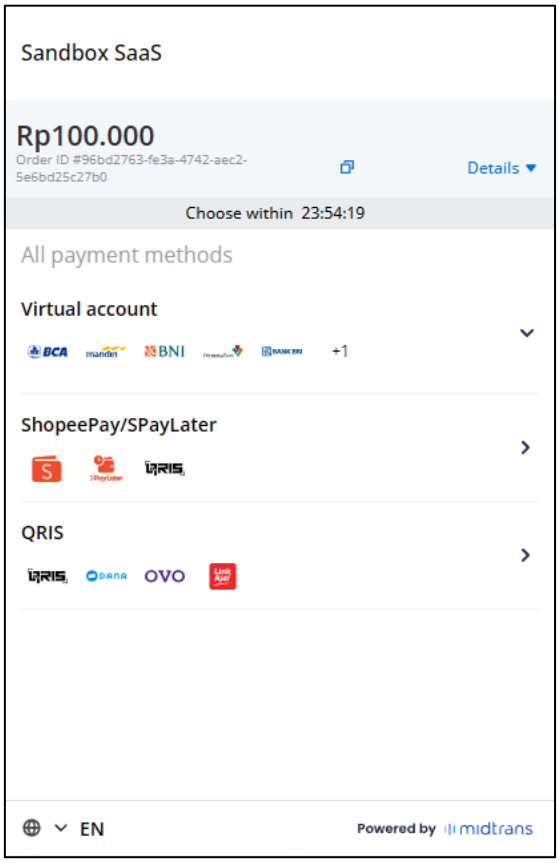
4.4.4 Hasil Pengujian Antarmuka Mengganti *Tier* Tenant

Hasil dari pengujian mengganti *tier* Tenant menggunakan antarmuka dipaparkan pada **Tabel 4.10**. Hasil pada tabel menunjukkan perbandingan kondisi Tenant yang berubah sebelum melakukan pergantian *tier*, dan setelah melakukan pergantian *tier*. Perpindahan *tier* akan menghasilkan pendireksian halaman ke laman pembayaran oleh Billing, yang ditunjukan pada Gambar 4.14.

Tabel 4.10 Hasil Pengujian P04, Status Tenant Menjadi Migrating

Kondisi Tenant Awal pada Antarmuka Beranda	

Kondisi Tenant Akhir pada Antarmuka Beranda	**Recently Added** Mahesa Todo - Saas Todo V2 Premium activated Decommission **All Tenants** Mahesa Todo - Saas Todo V2 Premium activated Decommission



Gambar 4.14 Hasil Pendireksian Laman Tenant ke Pembayaran

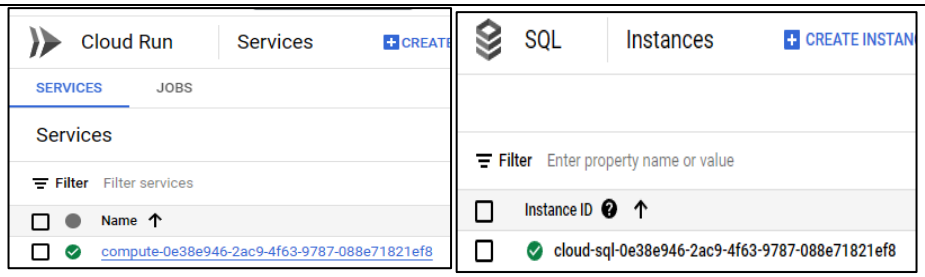
4.4.5 Hasil Pengujian Sistem Penghapusan Sumber Daya Tenant

Hasil *Logging server* yang didapat pada Gambar 4.15 menunjukkan bahwa proses penghapusan sumber daya sudah berhasil menghapus *instance cloud sql* yang merupakan basis data aplikasi Tenant; dan *instance cloud run* yang merupakan *instance backend* dari aplikasi Tenant. Perubahan keadaan yang disebabkan dari proses dekomisi Tenant dapat dilihat pada *cloud provider.* Dalam konteks tugas akhir ini, perubahan dilihat pada bagian daftar *cloud run* dan *cloud sql* di GCP. Output yang diinginkan adalah sumber daya yang dipakai oleh Tenant yaitu *instance* "cloud-run-0e38e946-2ac9-4f63-9787-088e71821ef8" dan *instance* "sql-0e38e946-2ac9-4f63-9787-088e71821ef8" akan menghilang dari daftar sumber daya GCP. Perbandingan kondisi awal dengan kondisi akhirnya dipaparkan dalam **Tabel 4.11**.

```
module.storage.google_sql_database_instance.storage: Still destroying... [id=cloud-sql-fc053939-721b-41d9-b721-
fbb870384d36, 1m1s elapsed]
module.storage.google_sql_database_instance.storage: Still destroying... [id=cloud-sql-fc053939-721b-41d9-b721-
fbb870384d36, 1m11s elapsed]
module.storage.google_sql_database_instance.storage: Still destroying... [id=cloud-sql-fc053939-721b-41d9-b721-
fbb870384d36, 1m21s elapsed]
module.storage.google_sql_database_instance.storage: Still destroying... [id=cloud-sql-fc053939-721b-41d9-b721-
fbb870384d36, 1m31s elapsed]

2024/07/13 10:26:03 ;1m/app/internal/infrastructure/postgres/postgres/tenant_repository.go:34 ;1mrecord not found
[2.726ms] ;1m[rows:0] SELECT * FROM "tenants" WHERE "id" = '1d807e03-88e0-403e-b42f-e5ab2c756547' LIMIT 1
data tenant dengan id 1d807e03-88e0-403e-b42f-e5ab2c756547 tidak ditemukan
module.storage.google_sql_database_instance.storage: Still destroying... [id=cloud-sql-fc053939-721b-41d9-b721-
fbb870384d36, 1m41s elapsed]
module.storage.google_sql_database_instance.storage: Still destroying... [id=cloud-sql-fc053939-721b-41d9-b721-
fbb870384d36, 1m51s elapsed]
module.storage.google_sql_database_instance.storage: Destruction complete after 1m57s

Destroy complete! Resources: 2 destroyed.
```

Gambar 4.15 Log Proses Penghapusan Sumber Daya di Cloud Provider

Tabel 4.11 Perbandingan Kondisi pada Cloud Provider saat Proses Penghapusan Sumber Daya

Kondisi awal *cloud run* dan *cloud sql* pada *cloud provider*	
Kondisi akhir *cloud run* dan *cloud sql* pada *cloud provider*	

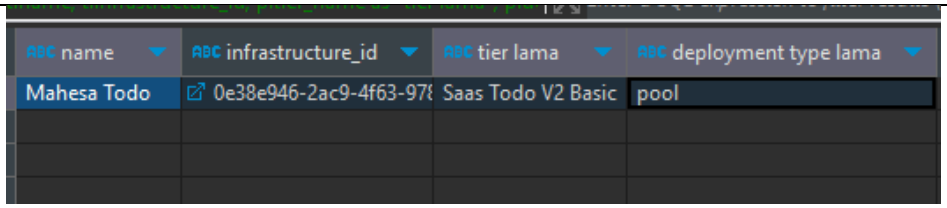
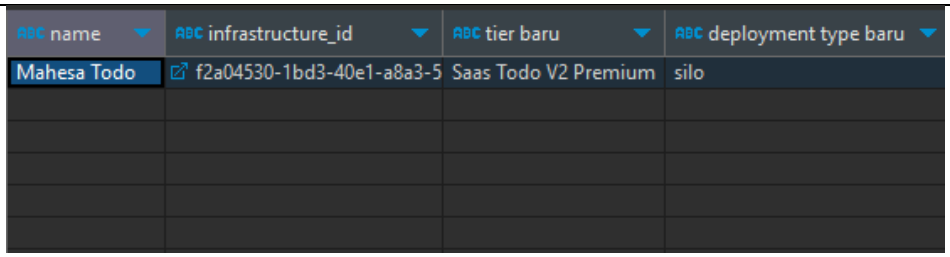
4.4.6 Hasil Pengujian Sistem Migrasi Sumber Daya Tenant dari POOL ke SILO

Hasil *logging* dari proses migrasi sumber daya Tenant dari POOL ke SILO ditunjukan dalam Gambar 4.16 yang menunjukkan bahwa sistem telah sukses melakukan migrasi sumber daya Tenant; Setelah melakukan proses migrasi, modul Tenant Management akan menerbitkan event "tenant_migrated" secara otomatis. Event tersebut akan ditangkap oleh modul Tenant Management yang bertugas untuk memberikan data infrastruktur baru kepada tenant seperti yang ditunjukan dalam **Tabel 4.12**. Modul Identity Management kemudian akan menangkap *event* "tenant_migrated" dan akan mergistrasikan url sumber daya baru tenant. Setelah modul Identity Management selesai mergistrasikan url sumber daya baru ke *auth ptovider,* modul Identity Management akan menerbitkan Event "tenant_registered" yang akan ditangkap oleh modul Tenant Management untuk membuat tenant menjadi aktif. Perbandingan kondisi awal dan akhir tenant yang memerlihatkan *tier* tenant sebelum melakukan pergantian *tier* dan setelah selesai melakukan migrasi dapat dilihat dalam **Tabel 4.13**).

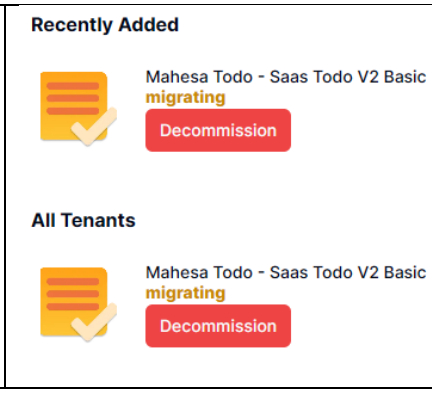
```
        "user": "default-f2a04530-1bd3-40e1-a8a3-5202709c5a19"
      }
    }
  },
  "resource_information": {
    "sensitive": false,
    "type": [
      "object",
      {
        "serving_url": "string"
      }
    ],
    "value": {
      "serving_url": "https://compute-f2a04530-1bd3-40e1-a8a3-5202709c5a19-lls7dcxv2q-et.a.run.app"
    }
  }
}
success melakukan migrasi
```


Gambar 4.16 Hasil Log Proses Migrasi dari Console Server

Tabel 4.12 Hasil Perbandingan Infrastruktur Tenant yang Lama Dengan yang Baru

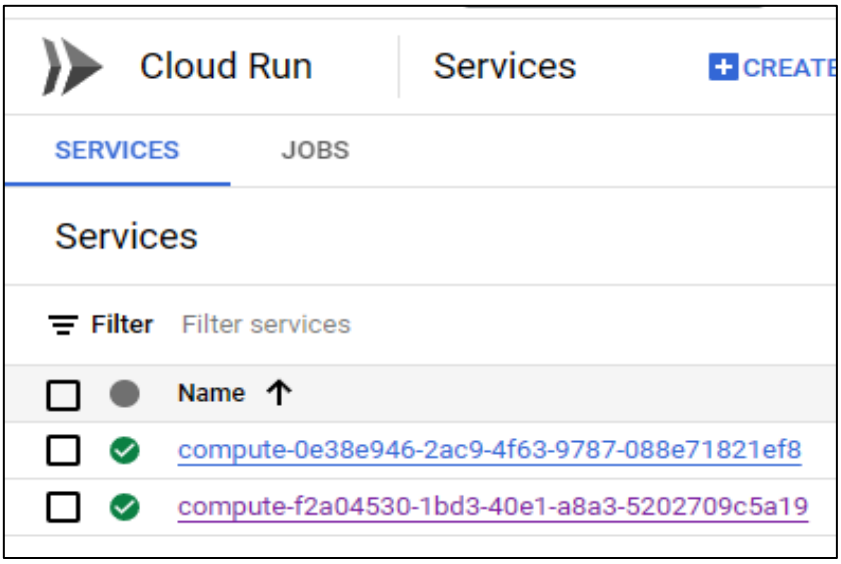
Infrastruktur Lama	
Infrastruktur Baru	

Tabel 4.13 Hasil Perbandingan Status Tenant pada Laman Beranda

Kondisi Tenant Awal pada Antarmuka Beranda	

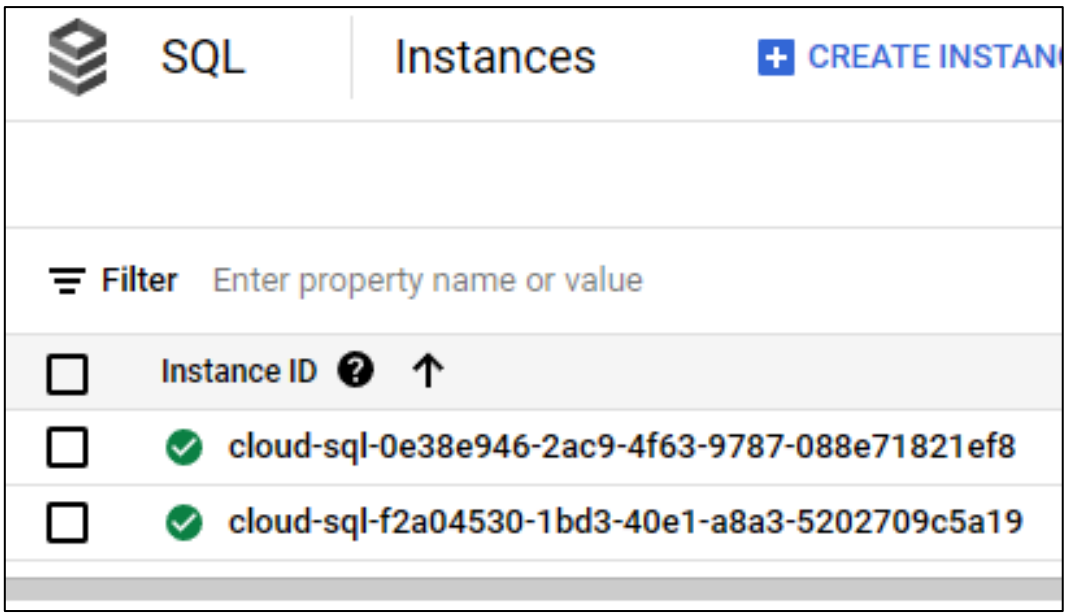
Kondisi Tenant Akhir pada Antarmuka Beranda	

Hasil *instance deployment* dapat dilihat pada panel Cloud Run dan SQL di GCP. Pada Gambar 4.17 diperlihatkan *instance POOL* yang awalnya dipakai tenant yaitu "compute- 0e38e946-2ac9-4f63-9787-088e71821ef8" yang mengarah ke id infrastruktur lama tenant. Lalu *instance SILO* baru yang digunakan tenant adalah "compute-f2a04530-1bd3-40e1-a8a3-5202709c5a19" juga sudah berhasil terdeploy. *Instance POOL* lama akan tetap tidak akan dihapus untuk beberapa lama karena dengan deployment tipe *POOL,* tenant bisa langsung mengambil slot untuk menggunakan *instance* yang sudah terdeploy, dengan begitu akan mempercepat lama waktu proses migrasi tenant dari *SILO* ke *POOL* yang akan dijelaskan pada subbab selanjutnya.



Gambar 4.17 Instance SILO yang Berhasil Terdeploy

Instance SQL masing-masing compute Cloud Run juga sudah berhasil terbuat seperti yang ditunjukan pada Gambar 4.18. Untuk mengecek apakah *instance* Cloud Run yang baru saja terdeploy bisa melakukan koneksi terhadap *instance* SQL yang baru saja terdeploy, dapat melihat bagian logs di konsol GCP. Gambar 4.19 menjelaskan di mana *instance* Cloud Run berhasil menjalankan servernya yang memiliki arti bahwa *instance* tersebut berhasil melakukan koneksi ke *instance* database SQL yang terdeploy. Jika *instance* Cloud Run tidak bisa melakukan koneksi ke *instance* database SQL, maka akan terlempar pesan error pada logging konsol GCP, dan server tidak akan bisa menyala.



Gambar 4.18 Instance Sql yang Berhasil Terdeploy

Gambar 4.19 Log Konsol GCP Menunjukan Instance Berhasil Berjalan

4.4.7 Hasil Pengujian Sistem Migrasi Sumber Daya Tenant dari SILO ke POOL

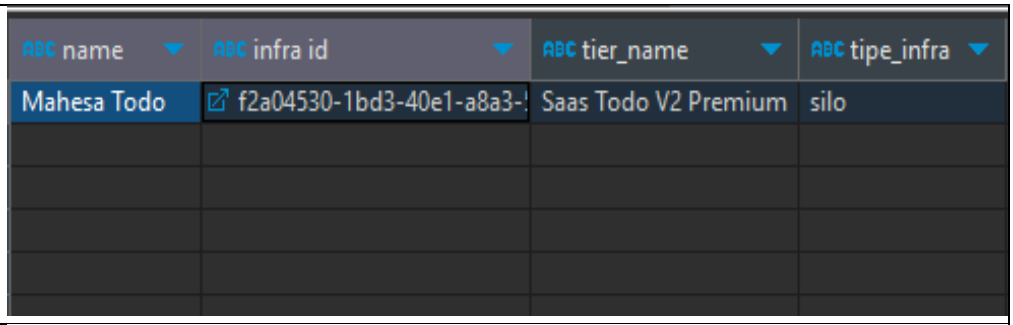
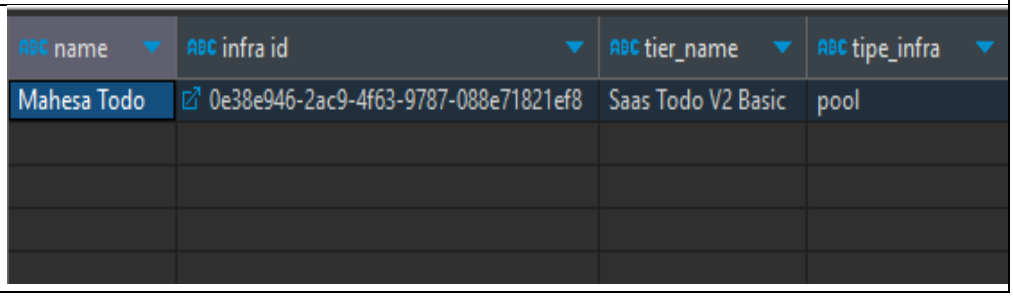
Hasil *logging* dari proses migrasi sumber daya Tenant dari SILO ke POOL ditunjukan dalam Gambar 4.20 yang menjelaskan bahwa proses migrasi sumber daya tenant berhasil dilakukan oleh sistem modul Tenant Management; Setelah melakukan proses migrasi, modul Tenant Management akan menerbitkan event “tenant_migrated” secara otomatis. Event tersebut akan ditangkap oleh modul Tenant Management yang bertugas untuk memberikan data infrastruktur baru kepada tenant seperti yang ditunjukan dalam **Tabel 4.14**, dan modul *Identity Management* yang akan mergistrasikan url sumber daya baru tenant dan menerbitkan Event “tenant_registered” yang akan membuat tenant menjadi aktif. Perbandingan kondisi awal dan akhir tenant yang memerlihatkan *tier* tenant sebelum melakukan pergantian *tier* dan setelah selesai melakukan migrasi dapat dilihat dalam

Tabel 4.15).

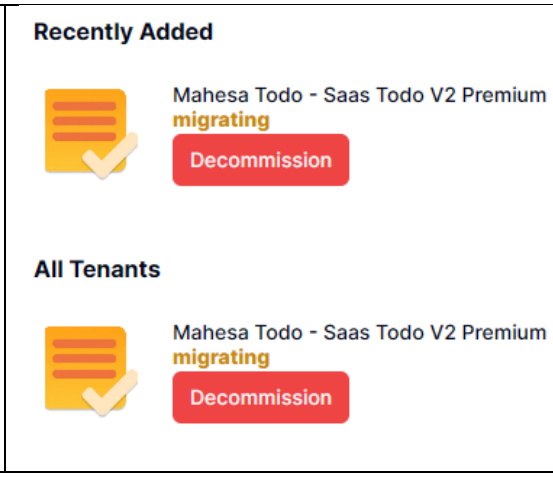
```
Compressing objects:  89% (113/126)
Compressing objects:  90% (114/126)
Compressing objects:  91% (115/126)
Compressing objects:  92% (116/126)
Compressing objects:  93% (118/126)
Compressing objects:  94% (119/126)
Compressing objects:  95% (120/126)
Compressing objects:  96% (121/126)
Compressing objects:  97% (123/126)
Compressing objects:  98% (124/126)
Compressing objects:  99% (125/126)
Compressing objects: 100% (126/126)
Compressing objects: 100% (126/126), done.
Total 238 (delta 66), reused 119 (delta 39), pack-reused 71
success melakukan migrasi
```

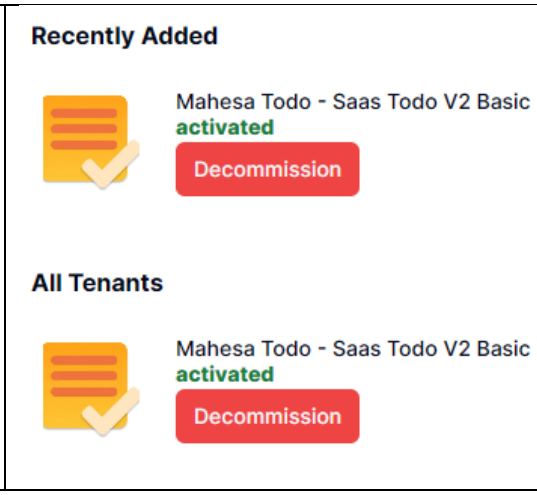
Gambar 4.20 Hasil Log Proses Migrasi dari Console Server

Tabel 4.14 Hasil Perbandingan Infrastruktur Tenant yang Lama Dengan yang Baru

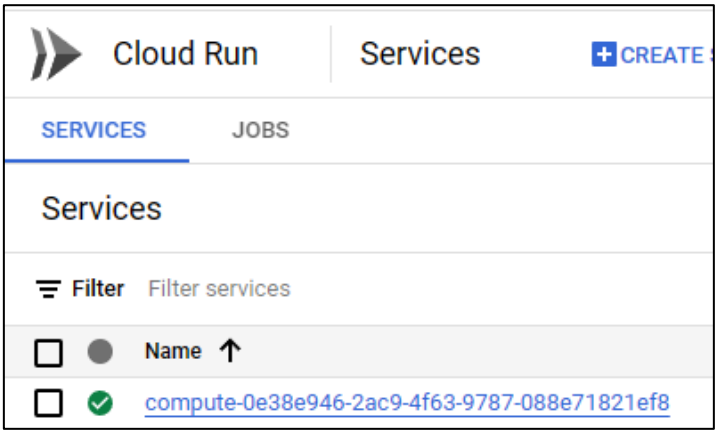
Infrastruktur Lama	
Infrastruktur Baru	

Tabel 4.15 Hasil Perbandingan Status Tenant pada Laman Beranda

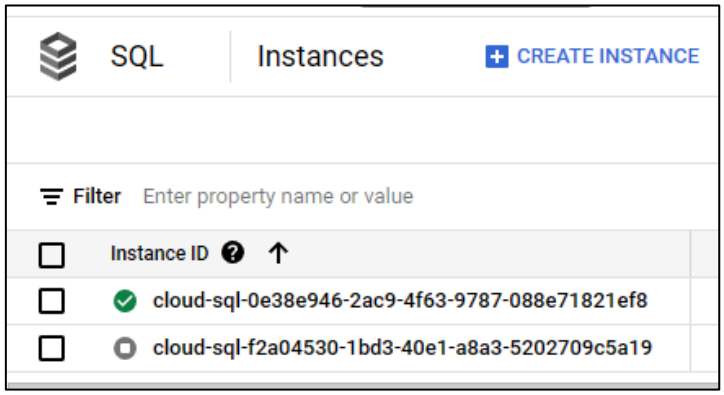
Kondisi Tenant Awal pada Antarmuka Beranda	

Kondisi Tenant Akhir pada Antarmuka Beranda	

Tenant yang melakukan migrasi sumber daya dari tipe *SILO* ke *POOL*, sumber daya *SILO* yang lama akan ditandai untuk dihapus secara periodik, dan sumber daya barunya langsung diarahkan ke *instance POOL* yang sudah pernah dideploy sebelumnya. *Instance SILO* SQL yang sudah ada akan diberhentikan untuk mengurangi biaya pemakaian dari GCP. Menggunakan metode ini, proses bisnis migrasi tenant akan berjalan semakin cepat, karena tidak melakukan deployment baru, melainkan menggunakan *instance POOL* yang sudah ada, dan langsung mengarahkan tenant ke *instance POOL* tersebut.



Gambar 4.21 Berhasil Menggunakan Instance POOL yang Sudah Ada



Gambar 4.22 Berhasil Menggunakan Instance SQL POOL yang Sudah Ada

4.4.8 Hasil Pengujian Refaktorisasi Kerangka Kerja Modul Tenant Management Tanpa Menggunakan Modul Billing dan IAM untuk Melakukan Migrasi Sumber Daya

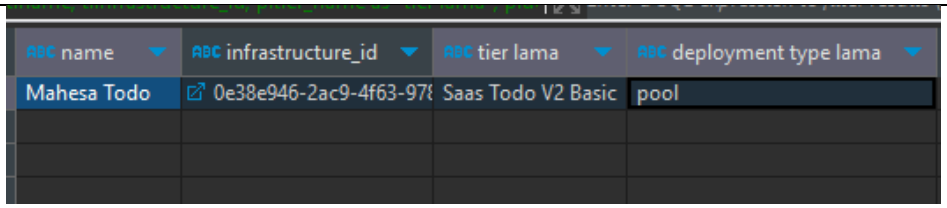
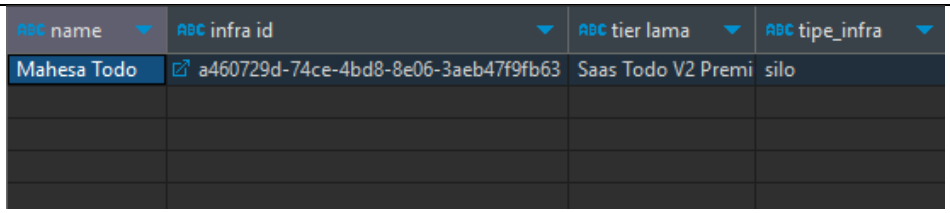
Setelah pengguna SaaS memilih *tier* tenant, sistem Tenant Management akan menerbitkan *event* "tenant_migrating_independently" sebagai pengganti *event* "billing_paid" dari modul Billing. Modul Tenant Management sendiri yang akan menangkap *event* "tenant_migrating_independently" dan akan langsung dilakukan migrasi seperti yang digambarkan pada Gambar 4.23


```
        "user": "default-d7d7eab1-6057-47c9-a2db-e3e95fd1d551"
      }
    }
  },
  "resource_information": {
    "sensitive": false,
    "type": [
      "object",
      {
        "serving_url": "string"
      }
    ],
    "value": {
      "serving_url": "https://compute-d7d7eab1-6057-47c9-a2db-e3e95fd1d551-lls7dcxv2q-et.a.run.app"
    }
  }
}
success melakukan migrasi
```

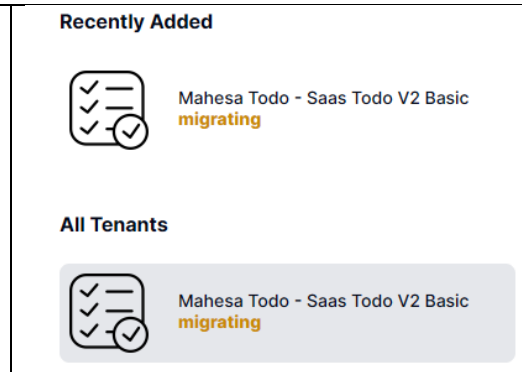
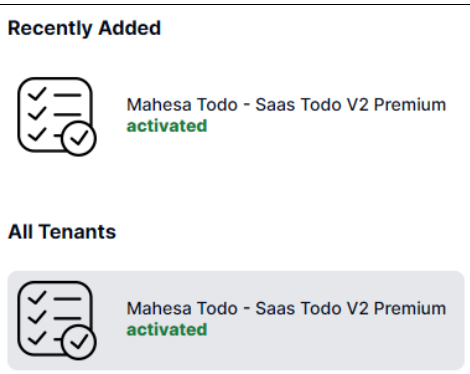

Gambar 4.23 Log Konsol Server Menampilkan Sukses Melakukan Migrasi

Setelah melakukan proses migrasi menggunakan Terraform, sistem akan langsung mendelegasikan infrastruktur baru ke tenant dan membuat tenant nya aktif tanpa menunggu *event* yang diterbitkan oleh modul IAM. Efek dari refaktorisasi adalah sumber daya tenant yang langsung dimigrasi *on-request* tenant mengubah *tier* tanpa *trigger* dari Billing dan statusnya yang langsung aktif tanpa *trigger* dari modul IAM. Hasil dari perbandingan data tenant yang ditampilkan dan di basis data dipaparkan dalam **Tabel 4.16** dan **Tabel 4.17**.

Tabel 4.16 Hasil Perbandingan Status Tenant pada Laman Beranda

Infrastruktur Lama	
Infrastruktur Baru	

Tabel 4.17 Hasil Perbandingan Status Tenant pada Laman Beranda

Kondisi Tenant Awal	
Kondisi Tenant Akhir	

4.5 Pembahasan

Setelah pengujian dilakukan dan hasil sudah didapatkan, berikut pembahasan kesimpulan dari hasil pengujian tersebut:

4.5.1 Pembahasan Pengujian Fungsional

Dari percobaan-percobaan yang telah dilakukan, didapat kesimpulan bahwa modul Tenant Management berhasil mengurus semua Usecase kebutuhan dan domain dari proses bisnis yang harus dapat dilakukan sebagai modul Tenant Management. Sistem Tenant Management juga berhasil untuk melakukan integrasi dengan modul eksternal seperti Billing dan IAM baik melalui panggilan API atau terbitan *Event*.

4.5.2 Pembahasan Pengujian Refaktorisasi

Pengujian refaktorisasi modul Tenant Management ditujukan untuk menilai desain dari rancang bangun modul Tenant Management yang telah dibuat. Desain yang kuat akan membuat modul mudah untuk dimodifikasi dengan usaha yang sedikit sesuai dengan kebutuhan bisnis. Sebagai contoh pada kasus pengujian yang telah dilakukan, ketika pengembang menginginkan modul Tenant Management berdiri sendiri tanpa menggunakan Billing, yang akan terjadi adalah proses migrasi sumber daya tidak perlu menunggu Billing untuk membuatkan penagihan. Modul Tenant Management akan menjadi pemicu utama dilakukannya migrasi sumber daya di *cloud provider*, perubahan yang diperlukan adalah menambah beberapa baris pada model dan *event listener*. Perubahan menambah baris tersebut mengikuti konsep *Clean Code Open-Closed Principle*, di mana kode sumber seharusnya terbuka untuk ditambahkan tetapi tertutup untuk dimodifikasi.

BAB V KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil selama proses perancangan dan pembangunan kerangka kerja pada tugas akhir, dapat diambil kesimpulan sebagai berikut:

1. Rancangan kerangka kerja modul Tenant Management menggunakan pola DDD pada arsitektur CQRS memudahkan pengembang untuk menggunakan kerangka kerja ini sebagai basis dalam membuat modul Tenant Management untuk kebutuhan bisnisnya. Inti proses bisnis yang sudah terabstraksikan dengan jelas dan implementasi kode yang didasari dengan antarmuka dari domain bisnis membuat kode mudah untuk diekstensikan sesuai kebutuhan pengembang.
2. Pembangunan kerangka kerja modul Tenant Mangement dilakukan menggunakan bahasa pemrograman Go untuk *backend* sebagai sistem deployment/migrasi dan penanganan *lifecycle* tenant. *Platform* GCP digunakan sebagai wadah pembangunan, pengaturan sumber daya tenant dan komunikasi antar modul.
3. Kerangka kerja modul Tenant Management diintegrasikan kepada aplikasi SaaS secara utuh dengan cara menggunakan *Event Driven Architecture* sebagai alat komunikasi antar modul yang difasilitasi oleh Pub/Sub dari *cloud provider* GCP. Event digunakan untuk melonggarkan keterikatan langsung modul Tenant Management dengan modul lain saat berintegrasi. Walaupun keterikatan langsung antar modul longgar, tetap ada keterikatan seperti event yang dipublish modul billing untuk memicu jalannya migrasi sumber daya. Walau begitu, modul Tenant Management tetap dapat berdiri sendiri dengan cara memodifikasi kode sumber dengan usaha minim karena penggunaan EDA dan DDD.
4. Pengujian fungsional kerangka kerja modul Tenant Management dilakukan menggunakan metode *black box* untuk mengevaluasi kebutuhan fungsional yang dapat dilakukan tenant. Pengujian refaktorisasi dilakukan dengan memodifikasi kode sumber modul untuk mengukur kemudahan mengembangkan modul Tenant Management dengan proses bisnis yang berbeda.

5.2 Saran

Modul Tenant Management masih bisa disempurnakan untuk meningkatkan pilihan provider dalam pengembangan aplikasi SaaS. Dukungan terhadap provider selain GCP dapat diimplementasikan seperti AWS atau Microsoft Azure

<halaman ini sengaja dikosongkan>

Daftar Pustaka

Alexander. (2024). *RANCANG BANGUN KERANGKA KERJA MODUL TENANT ONBOARDING UNTUK APLIKASI SOFTWARE AS A SERVICE*.

Amazon Web Services. (2024). *SaaS Architecture Fundamentals*.

Andrawos, M., & Helmich, M. (2017). *Cloud Native Programming with Golang: Develop microservice-based high performance web apps for the cloud with Go*. Packt Publishing Ltd.

Aung, T. H., Benlian, A., Hess, T., Buxmann, P., & Uddenberg, A. (2009). SaaS in Business: Exploring Strategic Benefits and Considerations of Software as a Service (SaaS) Model in Business Organizations. In *Business & Information Systems Engineering* (Vol. 1, Issue 5). https://doi.org/10.13140/2.1.3130.3043

Bezemer, C.-P., & Zaidman, A. (n.d.). *Challenges of Reengineering into Multi-Tenant SaaS Applications*.

Bezemer, C.-P., & Zaidman, A. (2010). Multi-tenant SaaS applications: Maintenance Dream or Nightmare? *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, 88–92. https://doi.org/10.1145/1862372.1862393

Cai, H., Reinwald, B., Wang, N., & Guo, C. J. (2013). SaaS Multi-Tenancy. In *Cloud Computing Advancements in Design, Implementation, and Technologies* (pp. 67–82). IGI Global. https://doi.org/10.4018/978-1-4666-1879-4.ch005

Donovan, A., & Kernighan, B. (2015). *The Go Programming Language*.

Erb, B., & Kargl, F. (2014). Combining Discrete Event Simulations and Event Sourcing. *Proceedings of the Seventh International Conference on Simulation Tools and Techniques*. https://doi.org/10.4108/icst.simutools.2014.254624

Golding, T. (2024). *Building Multi-Tenant SaaS Architectures*. O'Reilly Media, Inc.

Guo, C. J., Sun, W., Huang, Y., Wang, Z. H., & Gao, B. (2007). A Framework for Native Multi-Tenancy Application Development and Management. *The 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007)*, 551–558. https://doi.org/10.1109/CEC-EEE.2007.4

HashiCorp. (n.d.). *What is Terraform?* https://developer.hashicorp.com/terraform/intro

Hilmi, R. (2024). *RANCANG BANGUN KERANGKA KERJA MODUL IDENTITY ACCESS MANAGEMENT UNTUK APLIKASI SOFTWARE AS A SERVICE*.

Khononov, V. (2021). *Learning Domain-Driven Design*.

Korkmaz, N., & Nilsson, M. (2014). *Practitioners' view on command query responsibility segregation.*

Krebs, R., Momm, C., & Kounev, S. (2012). ARCHITECTURAL CONCERNS IN MULTI-TENANT SaaS APPLICATIONS. *Proceedings of the 2nd International Conference on Cloud Computing and Services Science*, 426–431. https://doi.org/10.5220/0003957604260431

Kulkarni, G., Shelke, R., Palwe, R., Khatawkar, P., Bhuse, S., & Bankar, H. (2013). Multi-tenant SaaS Cloud. *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, 1–4. https://doi.org/10.1109/ICCCNT.2013.6726487

Li, X. H., Liu, T. C., Li, Y., & Chen, Y. (2008). *SPIN: Service Performance Isolation Infrastructure in Multi-tenancy Environment* (pp. 649–663). https://doi.org/10.1007/978-3-540-89652-4_58

Martin, R. C. (2006). *Agile Principles, Patterns, and Practices in C#.*

Momm, C., & Krebs, R. (2011). *A Qualitative Discussion of Different Approaches for Implementing Multi-Tenant SaaS Offerings*.

Morakos, P., & Meliones, A. (2014). Design and implementation of a Cloud SaaS framework for Multi-Tenant applications. *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*, 273–278. https://doi.org/10.1109/IISA.2014.6878755

Rachmat, R. (2024). *RANCANG BANGUN KERANGKA KERJA MODUL BILLING UNTUK APLIKASI SOFTWARE AS A SERVICE*.

Rajković, P., Janković, D., & Milenković, A. (2013). *Using cqrs pattern for improving performances in medical information systems.*

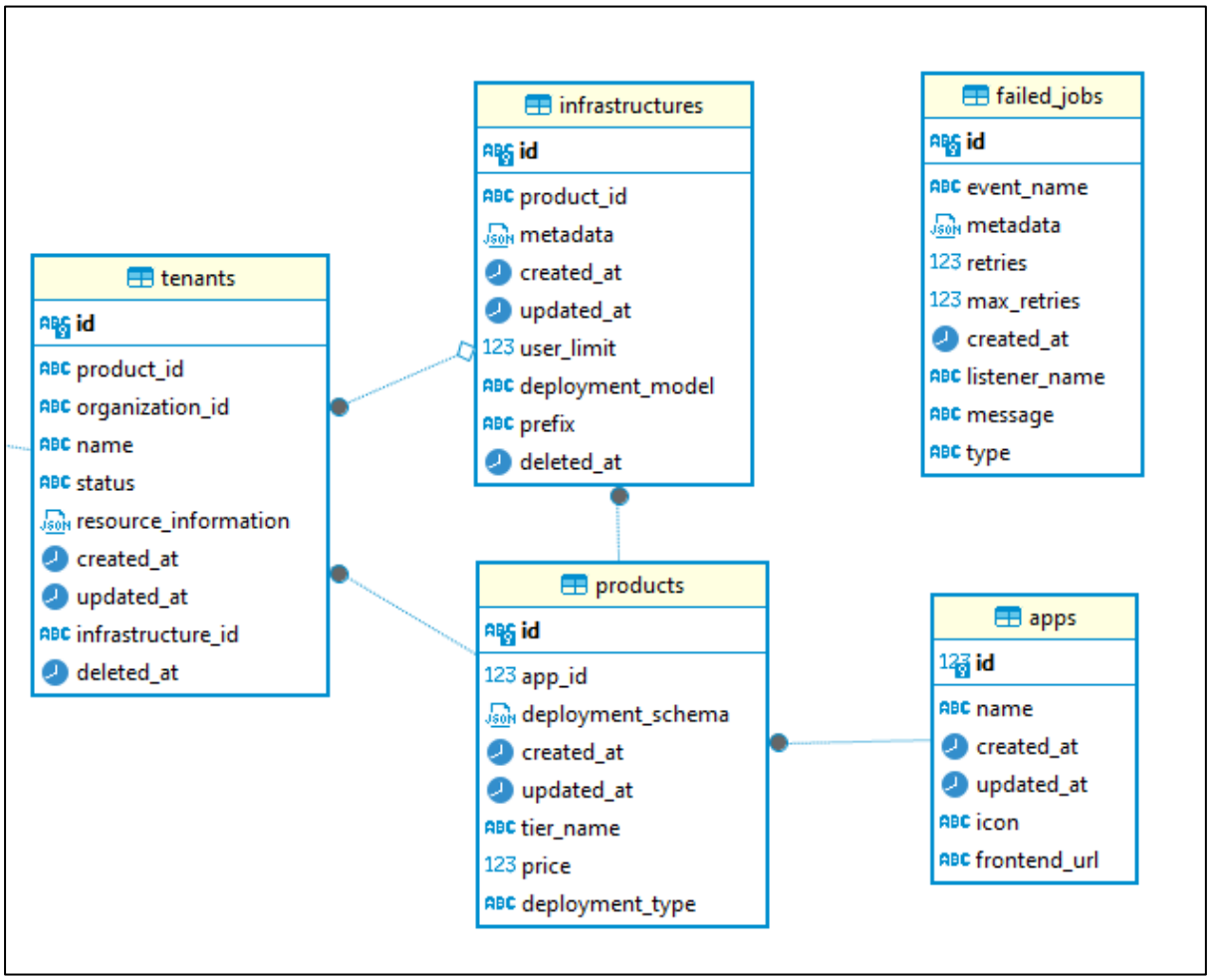
Stack, M. (2022). *Event-Driven Architecture in Golang*.

Vercel Inc. (n.d.). *What is Next.js?* https://nextjs.org/docs

Wankhede, P., Talati, M., & Chinchamalatpure, R. (2020). COMPARATIVE STUDY OF CLOUD PLATFORMS -MICROSOFT AZURE, GOOGLE CLOUD PLATFORM AND AMAZON EC2. *Journal of Research in Engineering and Applied Sciences*, *05*(02), 60–64. https://doi.org/10.46565/jreas.2020.v05i02.004

Zheng, J., & Du, W. (2014). *Cloud Services for Deploying Client-Server Applications to SaaS* (pp. 313–324). https://doi.org/10.1007/978-3-319-11167-4_31

LAMPIRAN



Lampiran 1 ER Diagram Modul Tenant Management

```
01    func (s TerraformService) MigrateTenantToTargetProduct(ctx context.Context, tenant *Tenant.Tenant,
target_product *Product.Product) error {
02        product_conf, err := s.constructTfProductConfig(target_product)
03        if err != nil {
04           return fmt.Errorf("gagal membangun konfigurasi product: %w", err)
05        }
06
07        tf, err := terraform.NewWorkspace(
08           os.Getenv("TF_WORKDIR"), os.Getenv("TF_EXECUTABLE"),
09           terraform_tenant.New(tenant.TenantId.String()),
10           terraform_product.UsingGit(product_conf),
11        )
12        if err != nil {
13           return fmt.Errorf("terjasi kesalahan dalam memroses executable terraform: %w", err)
14        }
15        defer tf.RemoveTenantDir()
16
17        old_infrastructure, err := s.infra_repo.Find(tenant.InfrastructureId)
18        if err != nil {
19           return fmt.Errorf("gagal mengambil data infrastructure tenant: %w", err)
20        }
21        defer s.CleanUpOldInfrastructure(old_infrastructure)
22
23        var infra_to_use *Infrastructure.Infrastructure
24        var new_metadata []byte
25
26        switch target_product.DeploymentType {
27        case terraform.POOL:
28           infra_to_use, err = s.infra_repo.FindAvailablePoolForProduct(target_product.ProductId)
29           if err != nil {
30              return err
31           }
32           if infra_to_use != nil {
33              err = tf.Migrate(ctx, old_infrastructure.Metadata, infra_to_use.Metadata)
34              if err != nil {
35                 return fmt.Errorf("gagal melakukan migrasi tenant %w", err)
36              }
37              return s.postMigration(infra_to_use, tenant)
38           }
39           infra_to_use = Infrastructure.CreatePoolConfig(target_product.ProductId)
40        case terraform.SILO:
41           infra_to_use = Infrastructure.CreateSiloConfig(target_product.ProductId)
42        }
43
44        tf.Tf_tenant.TenantEnv = append(tf.Tf_tenant.TenantEnv,
45           tfexec.Var(fmt.Sprintf("infrastructure_id=%s", infra_to_use.InfrastructureId.String())),
46           tfexec.Var(fmt.Sprintf("provider_id=%s", os.Getenv("GOOGLE_PROJECT_ID"))),
47        )
48        if os.Getenv("GOOGLE_CREDS_PATH") != "" {
49           tf.Tf_tenant.TenantEnv = append(tf.Tf_tenant.TenantEnv,
50              tfexec.Var(fmt.Sprintf("credentials=%s", os.Getenv("GOOGLE_CREDS_PATH"))),
51           )
52        }
53        tf.UseBackend(gcp.Backend(os.Getenv("GOOGLE_BUCKET"), infra_to_use.Prefix))
54
55        // di bawah ini konteksnya adalah ketika belum ada deployment tenant yang up
56        // dengan begitu lakukan Deploy dan Migrate
57        err = tf.Deploy(ctx)
58        if err != nil {
59           return fmt.Errorf("kegagalan dalam melakukan deployment tenant: %w", err)
60        }
61
62        new_metadata, err = tf.GetMetadata(ctx)
63        if err != nil {
64           return fmt.Errorf("kegagalan dalam mengambil metadata deployment")
65        }
66
67        infra_to_use.Metadata = new_metadata
68        err = tf.Migrate(ctx, old_infrastructure.Metadata, infra_to_use.Metadata)
69        if err != nil {
70           return fmt.Errorf("gagal melakukan migrasi tenant %w", err)
71        }
72
73        return s.postMigration(infra_to_use, tenant)
74    }
```

Lampiran 2 fungsi utama melakukan migrasi sumber daya menggunakan Terraform

```
01    func (s TerraformService) DecommissionInfrastructure(ctx context.Context,
infra *Infrastructure.Infrastructure) error {
02        product, err := s.product_repo.Find(infra.ProductId)
03        if err != nil {
04           return fmt.Errorf("terjadi kesalahan mengambil data product: %w", err)
05        }
06        if product == nil {
07           return fmt.Errorf(
                  "product dengan id %s tidak ditemukan", infra.ProductId.String())
08        }
09
10        product_conf, err := s.constructTfProductConfig(product)
11        if err != nil {
12           return fmt.Errorf("gagal membangun konfigurasi product: %w", err)
13        }
14
15        tf, err := terraform.NewWorkspace(
16           os.Getenv("TF_WORKDIR"), os.Getenv("TF_EXECUTABLE"),
17           terraform_tenant.New(uuid.NewString()),
18           terraform_product.UsingGit(product_conf),
19        )
20        if err != nil {
21           return fmt.Errorf(
                  "terjadi kesalahan dalam memroses executable terraform: %w", err)
22        }
23        tf.Tf_tenant.TenantEnv = append(tf.Tf_tenant.TenantEnv,
24           tfexec.Var(fmt.Sprintf("infrastructure_id=%s", infra.InfrastructureId.String())),
25           tfexec.Var(fmt.Sprintf("provider_id=%s", os.Getenv("GOOGLE_PROJECT_ID"))),
26        )
27        if os.Getenv("GOOGLE_CREDS_PATH") != "" {
28           tf.Tf_tenant.TenantEnv = append(tf.Tf_tenant.TenantEnv,
29              tfexec.Var(fmt.Sprintf("credentials=%s", os.Getenv("GOOGLE_CREDS_PATH"))),
30           )
31        }
32        tf.UseBackend(gcp.Backend(os.Getenv("GOOGLE_BUCKET"), infra.Prefix))
33
34        switch infra.DeploymentModel {
35        case terraform.SILO:
36           err = tf.Destroy(ctx)
37        case terraform.POOL:
38           if infra.UserCount > 0 {
39              return nil
40           }
41           err = tf.Destroy(ctx)
42        default:
43           err = fmt.Errorf(
                  "tipe deployment %s belum dapat dihandle", infra.DeploymentModel)
44        }
45        if err != nil {
46           return err
47        }
48        infra.Delete()
49        return s.infra_repo.Persist(infra)
50    }
```

Lampiran 3 fungsi utama melakukan dekomisi sumber daya tenant

```
func (s TerraformService) cleanUpOldInfrastructure(infra *Infrastructure.Infrastructure)
{
    // kalo pool, pool nya bisa dipake lagi,
    // walaupun pool nya kosong yo ndak papa
    if infra.DeploymentModel == terraform.SILO {
       s.event_service.Dispatch(
           events.INFRASTRUCTURE_DESTROYED,
               events.NewInfrastructureDestroyed(infra.InfrastructureId.String()))
    }
}
```

Lampiran 4 fungsi pembantu publish event untuk memicu dekomisi sumber daya tenant

```
func (s TerraformService) postMigration(infra *Infrastructure.Infrastructure,
tenant *Tenant.Tenant) error {
     if err := s.infra_repo.Persist(infra); err != nil {
        return fmt.Errorf(
             "gagal dalam melakukan persistansi data infrastruktur %s : %w",
                      infra.InfrastructureId.String(), err)
     }
     s.event_service.Dispatch(
             events.TENANT_MIGRATED,
                      events.NewTenantDelegatedToNewInfrastructure(
        tenant.TenantId.String(), infra.InfrastructureId.String(),
              infra.Metadata,
     ))
     return nil
}
```

Lampiran 5 fungsi pembantu setelah migrasi menyimpan data infrastruktur dan memicu event

BIODATA PENULIS

Penulis dilahirkan di Bekasi, 17 Oktober 2002, merupakan anak kedua dari 2 bersaudara. Penulis telah menempuh pendidikan formal yaitu di TK – SD – SMP Marsudirini Bekasi dan SMAS DeBritto Yogyakarta. Setelah lulus dari SMAS tahun 2020, Penulis diterima di Departemen Teknik Informatika FTEIC - ITS dan terdaftar dengan NRP 5025201105.

Di Departemen Teknik Informatika Penulis sempat aktif di beberapa kegiatan proyek seperti PPDB Surabaya dan Pemilihan Rektor ITS; kegiatan magang seperti bekerja di Simplus dan Direktorat Pengembangan Teknologi dan Sistem Informasi ITS; Dan kegiatan kepanitiaan seperti Schematics dan ILITS.